MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MÁSTER

Desarrollo de una aplicación de Realidad Aumentada para el apoyo en la toma de decisiones agronómicas con integración de un modelo de Inteligencia Artificial

Autor: Daniel Arqués Rubió

Director: José María Bengochea Guevara

Co-Director: Sergio Altares López

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo de una aplicación de Realidad Aumentada para el apoyo en la toma de decisiones agronómicas con integración de un modelo de Inteligencia Artificial

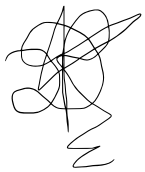
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Daniel Arqués Rubió Fecha: 28/06/2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

BENGOCHEA GUEVARA
JOSE MARIA - 50748973W

Fdo.: José María Bengochea Guevara Fecha: 28/06/2025

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Daniel Arqués Rubió DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Desarrollo de una aplicación de Realidad Aumentada para el apoyo en la toma de decisiones agronómicas con integración de un modelo de Inteligencia Artificial, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

de derechos derivada de las obras objeto de la cesión.

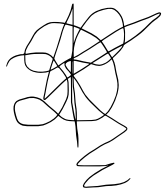
6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 28 de junio de 2025

ACEPTA



Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MÁSTER

Desarrollo de una aplicación de Realidad Aumentada para el apoyo en la toma de decisiones agronómicas con integración de un modelo de Inteligencia Artificial

Autor: Daniel Arqués Rubió

Director: José María Bengochea Guevara

Co-Director: Sergio Altares López

Madrid, 2025

Agradecimientos

Muchas gracias a Chema y a Sergio por su apoyo y dedicación con este proyecto.

Con esto pongo fin a mi etapa universitaria después de seis años de esfuerzo y trabajo duro. Estoy muy agradecido de haber vivido tantos momentos buenos, y otros que no tanto. Durante este tiempo he podido conocer tres ciudades increíbles y, en ellas, a personas también increíbles. Me gustaría agradecer a mis amigos y compañeros de clase por haber compartido tanto; nada de esto habría sido lo mismo sin vosotros. Quiero agradecer especialmente a Sara; eres, sin ninguna duda, lo mejor que me han dado estos años universitarios.

También quiero agradecer a mi familia, los que siempre han estado a mi lado incondicionalmente, gracias por creer siempre en mí. Un recuerdo especial para mi abuela Maria Carme; al final tantos números no me han dado dolor de cabeza. Ojalá pudiera compartir todo esto contigo.

Finalmente, a mis padres y a mi hermana Maria. Sobran las palabras. *Us estimo molt*.

DESARROLLO DE UNA APLICACIÓN DE REALIDAD AUMENTADA PARA EL APOYO EN LA TOMA DE DECISIONES AGRONÓMICAS CON INTEGRACIÓN DE UN MODELO DE INTELIGENCIA ARTIFICIAL

Autor: Arqués Rubió, Daniel.
Director: Bengochea Guevara, José María.
Codirector: Altares López, Sergio.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto presenta una herramienta de apoyo agronómico basada en inteligencia artificial y realidad aumentada, orientada a mejorar la toma de decisiones en la gestión del viñedo. Se ha desarrollado una aplicación móvil capaz de mostrar información sobre las parcelas, como las tareas pendientes o los rendimientos, basándose en su ubicación GNSS y detectar enfermedades en tiempo real mediante una red neuronal convolucional, integrada en una aplicación funcional para dispositivos Apple y Android.

Palabras clave: Realidad Aumentada, Inteligencia Artificial, Agricultura de Precisión, Enfermedades de la vid, Geolocalización

1. Introducción

La transformación digital en el sector agrario está dando lugar a nuevas herramientas capaces de optimizar la gestión de cultivos y mejorar la eficiencia en la toma de decisiones. En este contexto, la realidad aumentada combinada con modelos de inteligencia artificial permite ofrecer al agricultor una nueva forma de interpretar la información del campo directamente sobre el terreno. En particular, las explotaciones vitivinícolas presentan una alta complejidad en el seguimiento de tareas, diagnóstico fitosanitario y gestión de las parcelas.

2. Definición del proyecto

El objetivo principal del proyecto es diseñar, implementar y validar una aplicación multiplataforma (Android/iOS) que facilite el día a día del agricultor en el campo. Esta solución permite al agricultor conocer en tiempo real y en función de su ubicación GNSS, qué tareas están pendientes en la parcela donde se encuentra, acceder al histórico de rendimientos y realizar un diagnóstico de la vid mediante inteligencia artificial para identificar las enfermedades más frecuentes. La aplicación se comunica con un servidor remoto que procesa las imágenes capturadas y devuelve un resultado visual superpuesto mediante realidad aumentada.

3. Descripción del sistema

El sistema desarrollado está compuesto por tres componentes principales. El primero es una aplicación móvil desarrollada en Unity, equipada con módulos de realidad aumentada y acceso a sensores GNSS y cámara, que muestran información de la parcela actual, así como de las tareas y rendimientos. En segundo lugar, un servidor desarrollado en Python encargado de recibir imágenes, procesarlas y ejecutar el modelo de

inteligencia artificial. Finalmente, una red neuronal convolucional entrenada para clasificar enfermedades comunes de la vid a partir de imágenes reales captadas en campo.

El flujo de funcionamiento es el siguiente: el usuario activa la app, que identifica la parcela mediante GNSS. Una vez allí, puede consultar tareas y rendimientos asociados a esa zona y tomar fotografías de la vid que se envían automáticamente al servidor. Tras el procesamiento, el diagnóstico es mostrado en la aplicación.

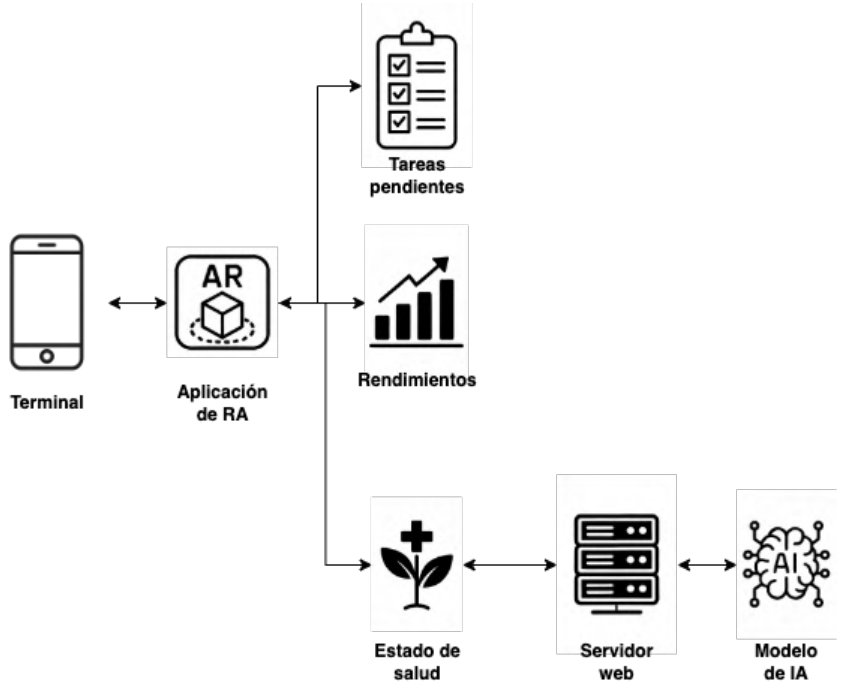


Figura 1 Flujo del sistema implementado.

4. Resultados

La aplicación ha sido probada en tres parcelas reales con distinta localización geográfica y características agronómicas, y se ha logrado una correcta detección de la parcela, así como una correcta visualización de los rendimientos y tareas asociados. Por otro lado, el sistema de diagnóstico de vides ha alcanzado una precisión superior al 95% en la detección de enfermedades en imágenes reales de validación (con las que el modelo no ha sido entrenado).

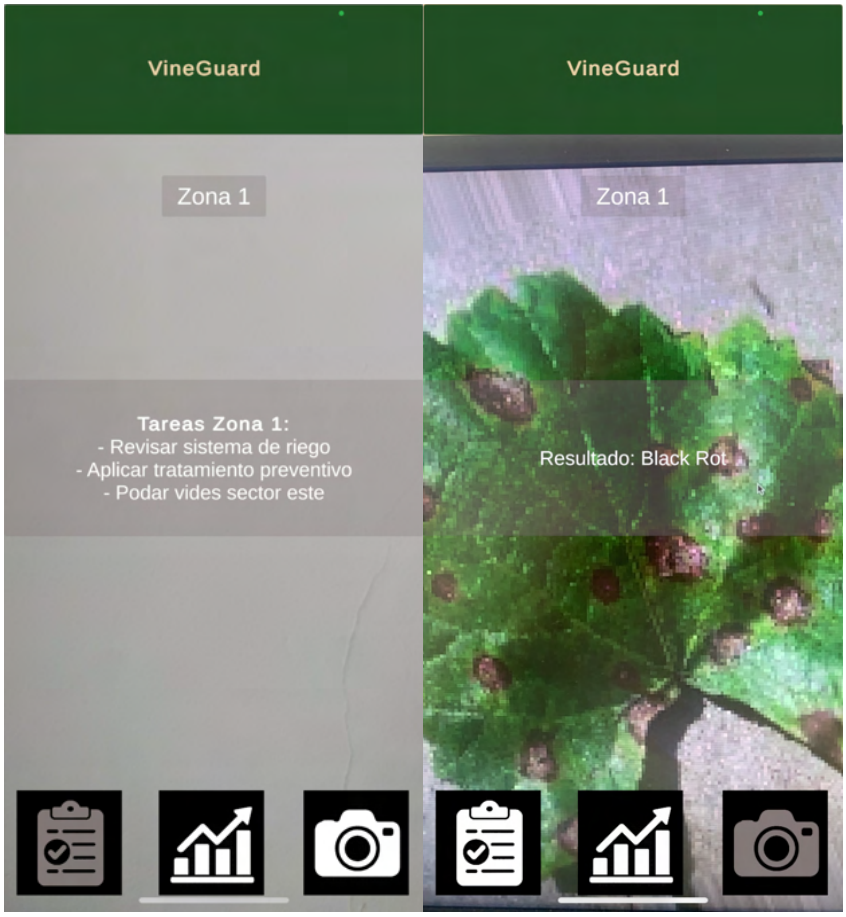


Figura 2 Identificación de parcela y de tareas asociadas y predicción correcta de enfermedad.

5. Conclusiones

El presente proyecto demuestra que la integración de tecnologías emergentes como la realidad aumentada y la inteligencia artificial puede ser aplicada con éxito al sector agrícola, particularmente en la viticultura. La aplicación desarrollada proporciona un entorno intuitivo y eficaz para la toma de decisiones, mejorando la eficiencia operativa y permitiendo un control más preciso de las tareas y del estado de la vid.

DEVELOPMENT OF AN AUGMENTED REALITY APPLICATION TO SUPPORT AGRONOMIC DECISION-MAKING WITH INTEGRATION OF AN ARTIFICIAL INTELLIGENCE MODEL

Author: Arqués Rubió, Daniel.
Supervisor: Bengochea Guevara, José María.
Co-Supervisor: Altares López, Sergio
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project presents an agronomic support tool based on artificial intelligence and augmented reality, aimed at improving decision-making in vineyard management. A mobile application has been developed that is capable of displaying information about plots, such as pending tasks or yields, based on their GNSS location, and detecting diseases in real-time through a convolutional neural network, integrated into a functional application for Apple and Android devices.

Keywords: Augmented Reality, Artificial Intelligence, Precision Agriculture, Grapevine Diseases, Geolocation

1. Introduction

The digital transformation of the agricultural sector is giving rise to new tools capable of optimizing crop management and improving decision-making efficiency. In this context, augmented reality combined with artificial intelligence models offers farmers a new way to interpret field data directly on-site. Vineyards, in particular, present a high complexity in task tracking, phytosanitary diagnosis, and parcel management.

2. Project definition

The main goal of this project is to design, implement, and validate a cross-platform application (Android/iOS) that facilitates the daily work of farmers in the field. This solution enables the user to know, in real time and based on GNSS location, what tasks are pending in the parcel where they are located, to access the historical yield records, and to perform an AI-based diagnosis of the vine to identify the most common diseases. The application communicates with a remote server that processes the captured images and returns a visual result displayed through augmented reality.

3. Description of the system

The developed system consists of three main components. First, a mobile application developed in Unity, equipped with augmented reality modules and access to GNSS and camera sensors, displaying information about the current parcel, as well as associated tasks and yield data. Second, a Python-based server responsible for receiving images, processing them, and running the artificial intelligence model. Finally, a convolutional neural network trained to classify common grapevine diseases from real images taken in the field.

The operating flow is as follows: the user launches the app, which identifies the parcel via GNSS. Once there, they can view tasks and yields associated with that area and take photos of the vine, which are automatically sent to the server. After processing, the diagnosis is displayed in the application.

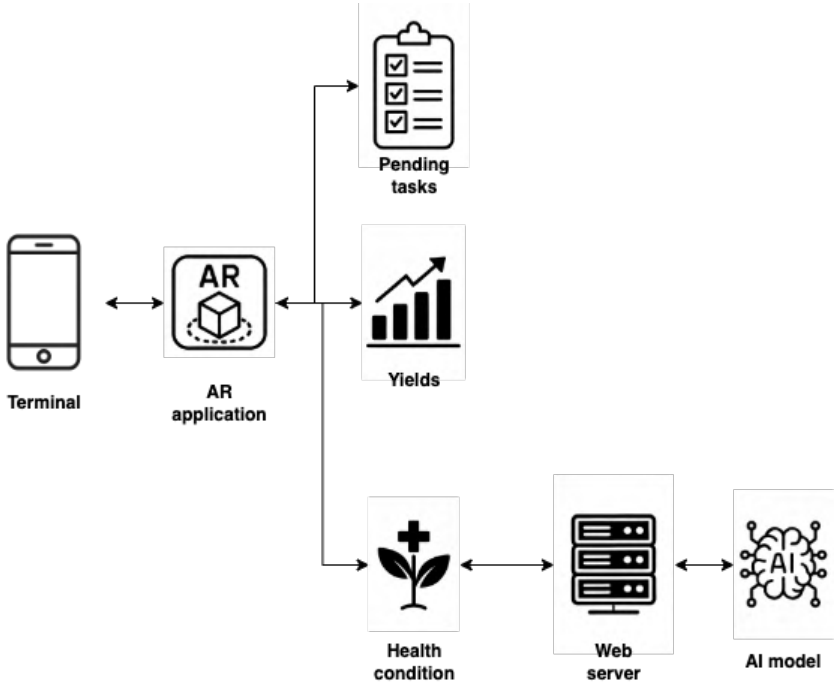


Figura 3 Flow of the implemented system.

4. Results

The application has been tested on three real vineyard plots with different geographic locations and agronomic characteristics. Correct parcel detection was achieved, along with proper visualization of associated yields and tasks. Additionally, the vine disease diagnosis system reached an accuracy above 95% when detecting diseases in real validation images that the model had not been trained on.

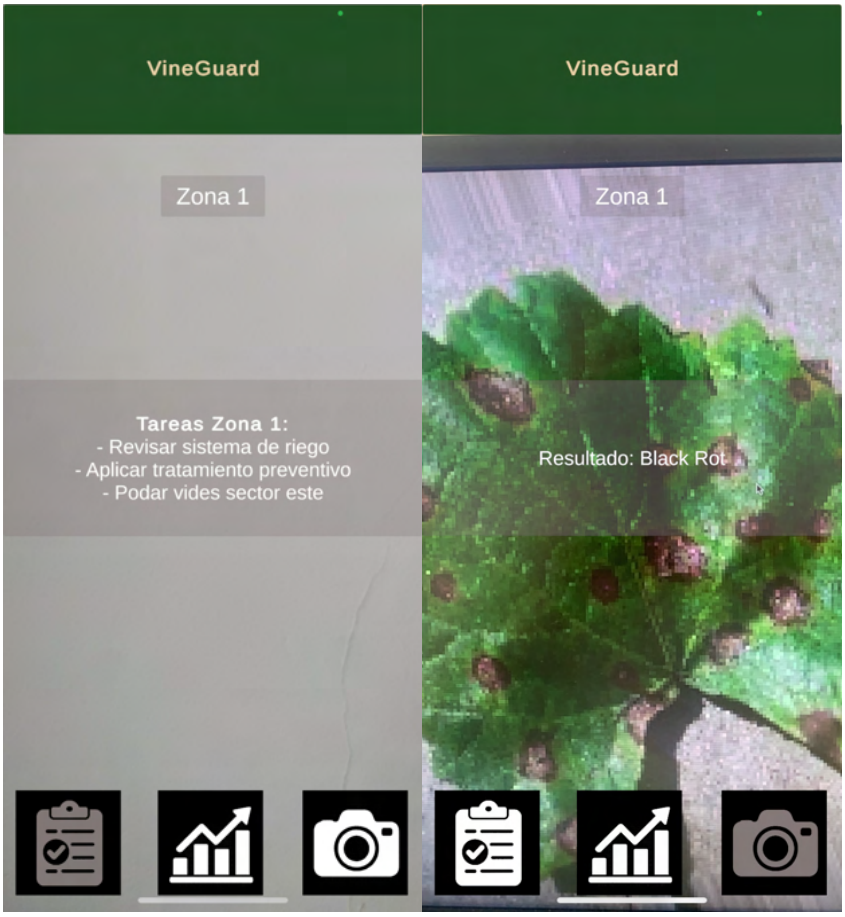


Figura 4 Parcel and associated tasks identification and correct disease prediction.

5. Conclusions

This project demonstrates that the integration of emerging technologies such as augmented reality and artificial intelligence can be successfully applied to the agricultural sector, particularly in viticulture. The developed application provides an intuitive and effective environment for decision-making, enhancing operational efficiency and enabling more precise control of both tasks and vine health.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

El presente proyecto surge de la necesidad de aplicar soluciones tecnológicas innovadoras al sector primario, concretamente al ámbito vitivinícola. Este sector, de gran peso económico, social y cultural en muchas regiones del mundo, y especialmente en España, se enfrenta actualmente a múltiples retos derivados de la necesidad de modernizar sus procesos, mejorar su productividad y adaptarse a un entorno cada vez más competitivo y exigente.

Tradicionalmente, la gestión de una explotación agraria vitivinícola ha estado marcada por procesos manuales, dispersos y poco sistematizados. Sin embargo, la digitalización del campo y la aplicación de tecnologías avanzadas como la inteligencia artificial o la realidad aumentada ofrecen hoy nuevas oportunidades para transformar profundamente este modelo. La combinación de dichas herramientas con el conocimiento agronómico tradicional puede permitir un control más preciso del viñedo, la optimización de los recursos y una toma de decisiones más informada y eficiente.

Este proyecto tiene como principal motivación facilitar y unificar las tareas diarias en el viñedo mediante el uso de tecnologías avanzadas, proporcionando también información contextualizada y en tiempo real de cada parcela. De este modo, se pretende mejorar el rendimiento operativo de los agricultores, reducir el margen de error humano y optimizar el seguimiento de cultivos y labores realizadas. Para ello, supone interesante el estudio de tecnologías de realidad aumentada que permitan combinar información con la realidad según la ubicación actual.

Adicionalmente, se pretende mejorar y automatizar la detección de posibles enfermedades en la vid a partir de un diagnóstico en tiempo real, contribuyendo así a una gestión preventiva más eficaz y a la reducción del uso innecesario de fitosanitarios. Una herramienta clave para alcanzar este objetivo es la inteligencia artificial, una tecnología en auge que ha demostrado ser válida en una diversidad de entornos.

En definitiva, este proyecto busca tender puentes entre la innovación tecnológica y uno de los sectores más tradicionales de nuestra economía, promoviendo una agricultura más inteligente, sostenible y competitiva.

Capítulo 2. ESTADO DEL ARTE

En este capítulo se van a analizar diferentes proyectos relacionados principalmente con el sector vitivinícola, la realidad aumentada (RA) y la inteligencia artificial (IA), tecnologías que pueden ser muy útiles para optimizar las labores del sector vitivinícola que se han explicado en la introducción.

En primer lugar, en la identificación de enfermedades mediante IA se encuentra la plataforma AIGrapeCare, que usa redes neuronales convolucionales para identificar enfermedades mediante imágenes de las hojas de vid (Elsherbiny et al., 2024). En este caso, se puede diagnosticar el estado con una precisión superior al 96,6%, diagnosticando 4 enfermedades o si la vid está sana. Sin embargo, el algoritmo utiliza solamente 296 imágenes, por lo que puede sufrir de sobreajuste o de baja capacidad de generalización.

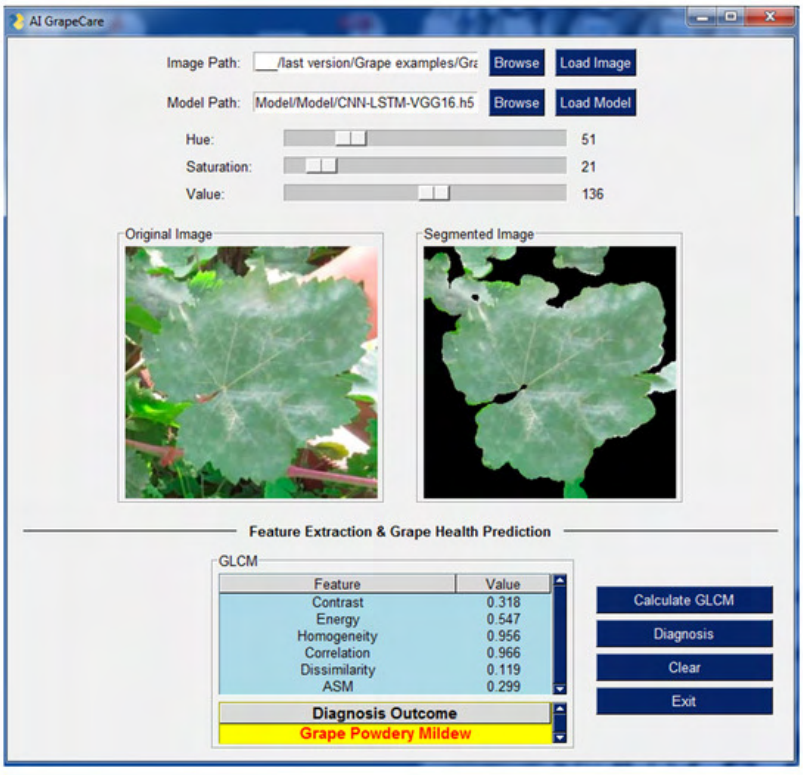


Figura 5 Ejemplo de la interfaz gráfica de AI GrapeCare (Elsherbiny et al., 2024).

Otro proyecto relacionado con la detección preventiva de enfermedades de la vid haciendo uso de modelos predictivos de IA es Viticast. En este caso, se trata de un grupo operativo supraautonómico que ha desarrollado un modelo predictivo que combina datos meteorológicos como la humedad, temperatura y precipitaciones junto con información fenológica del viñedo (las etapas del desarrollo de la vid) y con la concentración de esporas e inóculo (Piña-Rey et al., 2021). El objetivo es el de estimar con antelación la producción de cosecha y el de reducir los tratamientos antifúngicos que se aplican en el viñedo, para conseguir la elaboración de vinos de más calidad y que la producción sea más sostenible al minimizar la aplicación de fitosanitarios en el medio ambiente.

También se han realizado otros proyectos como GlobalViti con el objetivo de predecir enfermedades y de mitigar los efectos del cambio climático en la salud de la vid (GlobalViti, s.f.), o Televitis, un grupo de I+D+i de la Universidad de la Rioja que investigan en viticultura de precisión y en la aplicación de nuevas tecnologías al viñedo (Televitis, s.f.).

Por otro lado, en cuanto al estado de aplicaciones de RA para viñedos, destacan ejemplos como el de 3D2cut. Se trata de una *startup* suiza que ofrece unas gafas de realidad aumentada con un software basado en IA para digitalizar y facilitar la poda de vides, que es uno de los trabajos más costosos y que requieren más precisión a la hora de gestionar una vid. Las gafas son capaces de analizar imágenes de vides en tiempo real para recomendar los cortes óptimos (3D2cut, s.f.).

Figura 6 Gafas de RA de 3D2cut (3D2cut, s.f.).

En el marco de la innovación aplicada a la agricultura de precisión, el proyecto europeo FlexiGroBots (Flexible and Autonomous Robots for Smart Agriculture), financiado dentro del programa Horizonte 2020, representa una de las iniciativas más relevantes de los últimos años. Este proyecto, en el que participan instituciones y empresas punteras del ámbito de la robótica y la inteligencia artificial, tiene como objetivo el despliegue de flotas de robots autónomos para mejorar la eficiencia, sostenibilidad y precisión de los procesos agrícolas.

Uno de los pilotos más destacados del proyecto tuvo lugar en España, centrado específicamente en el cultivo de la vid. En este caso, se abordó el problema de la detección y tratamiento de la enfermedad causada por el hongo *Botrytis cinerea*, que afecta a la calidad y productividad de los viñedos (Ribeiro et al., 2023). La solución tecnológica propuesta consistía en un sistema de detección temprana basado en redes neuronales convolucionales entrenadas para identificar signos visuales de la enfermedad a partir de imágenes capturadas por vehículos aéreos no tripulados (UAVs) y vehículos terrestres autónomos (UGVs).

Una vez detectadas las zonas afectadas mediante el análisis de imágenes, se diseñó una estrategia de actuación localizada, aplicando fitosanitarios únicamente en las áreas

infectadas, con lo que se conseguía reducir el uso de productos químicos, minimizar el impacto ambiental y optimizar los recursos empleados en el tratamiento de las vides.

Finalmente, se analizará también el estado de la cuestión de aplicaciones de la industria agrícola que ofrezcan información sobre las parcelas mediante la localización GNSS. El Sistema Global de Navegación por Satélite (GNSS) es una tecnología de navegación y temporización a nivel global, y está compuesto por tres sistemas satelitales principales: GPS (Estados Unidos), GLONASS (Rusia) y Galileo (Europa) (Al-Bayari & Sadoun, 2007).

En este caso, es destacable el proyecto SIGPAC, el Sistema de Identificación de Parcelas Agrícolas implantado en toda la Unión Europea para la aplicación de las ayudas de la Política Agrícola Común (PAC) a los agricultores y ganaderos. Se trata de una base de datos con imágenes cartográficas digitalizadas, en este caso de toda España, con información y atributos de cada parcela. Esta información puede ser consultada a través de un visor cómo el mostrado en la figura 7, con diferentes opciones a elegir. Pese a ser concebido como un proyecto enfocado a la agricultura, hoy en día resulta también de gran utilidad en otros ámbitos como el urbanismo, las infraestructuras, la hidrología o la geología (Comunidad de Madrid, s.f.).

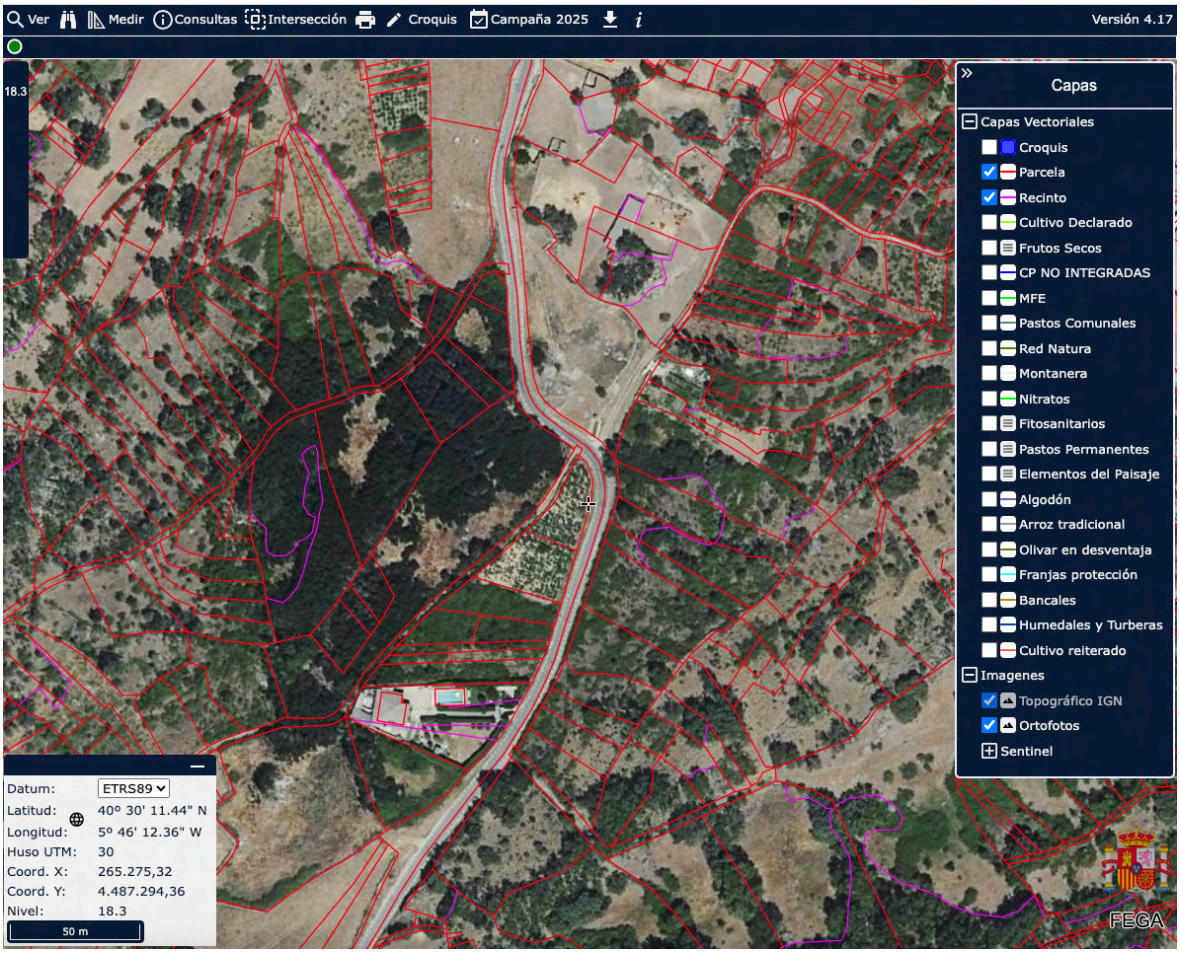


Figura 7 Visor SIGPAC. (SIGPAC, s.f.)

Capítulo 3. DEFINICIÓN DEL TRABAJO

3.1 JUSTIFICACIÓN

Dados los retos asociados a nivel global, en este trabajo nos centramos en el sector vitivinícola, el cual enfrenta actualmente numerosos desafíos derivados del acelerado cambio climático, de la necesidad de optimizar recursos y una creciente presión por reducir el uso de productos fitosanitarios, así como de una gran competitividad en el tejido empresarial. En este contexto, la incorporación de tecnologías digitales y de automatización representa no solo una oportunidad, sino una necesidad para avanzar hacia una agricultura más eficiente, sostenible y tecnológicamente avanzada.

A pesar del progreso en herramientas de agricultura de precisión, muchas explotaciones siguen basando decisiones cruciales en métodos tradicionales de observación y decisión, lo cual puede implicar una gestión reactiva e ineficiente del viñedo. Como se ha visto en el estado del arte, existen soluciones parciales como estaciones meteorológicas, sensores de suelo o imágenes satelitales, pero estas potentes soluciones no siempre se integran de forma práctica en el día a día del agricultor.

Este proyecto pretende aportar valor al proponer una solución integral y portátil basada en una aplicación de realidad aumentada y potenciada por inteligencia artificial, que permite al usuario identificar visualmente la parcela en la que se encuentra, consultar las tareas pendientes o los rendimientos y obtener un diagnóstico automático del estado sanitario de la vid, todo desde un dispositivo móvil convencional al alcance de cualquiera.

Desde el punto de vista técnico, la justificación de este trabajo radica en 3 pilares:

- La realidad aumentada permite visualizar información contextual de forma intuitiva y en tiempo real basándose en la geolocalización, y todo ello sin necesidad de equipos especializados.

- El uso de un modelo de IA entrenado con imágenes reales proporciona diagnósticos rápidos, que pueden mejorar la toma de decisiones agronómicas del sector vitivinícola. Gracias a la geolocalización de las plantas enfermas, los tratamientos se vuelven más eficientes, reduciendo el uso de fitosanitarios y promoviendo una agricultura de precisión más respetuosa con el medio ambiente.

- La solución propuesta es escalable, modular y fácilmente desplegable, por lo que puede adaptarse a otros cultivos o regiones sin un rediseño completo.

Finalmente, desde un punto de vista académico, este trabajo permite aplicar e integrar conocimientos avanzados en desarrollo multiplataforma, visión por computador, sistemas distribuidos, protocolos de comunicación y diseño de interfaces interactivas, lo que lo convierte en un caso práctico multidisciplinar con un alto valor formativo y técnico.

3.2 OBJETIVOS

El presente proyecto tiene como objetivo principal el desarrollo de una aplicación móvil multiplataforma basada en realidad aumentada, que sirva como herramienta de apoyo en la toma de decisiones agronómicas en explotaciones vitivinícolas. Esta solución pretende combinar la tecnología de RA con algoritmos de inteligencia artificial para proporcionar información en el campo, desde la visualización de tareas y rendimientos al diagnóstico del estado sanitario de las plantas de la vid.

De manera más concreta, entre los objetivos específicos del proyecto destacan:

- Diseñar e implementar una aplicación móvil compatible con Android y iOS, con una interfaz intuitiva y orientada al uso en el sector agrario.
- Integrar un sistema de posicionamiento GNSS para la identificación automática de parcelas y visualización de datos asociados.

- Incorporar un módulo de realidad aumentada que permita combinar el entorno con información como tareas pendientes de la parcela o históricos de producción.
- Desarrollar e integrar un modelo de inteligencia artificial capaz de identificar enfermedades comunes de la vid a partir de imágenes tomadas con la cámara del dispositivo.
- Implementar un servidor web ligero y funcional que reciba las imágenes desde la aplicación, procese los datos mediante el modelo entrenado y devuelva el diagnóstico al usuario de la aplicación.
- Validar la funcionalidad y precisión del sistema mediante pruebas.

Figura 8 Representación visual de los objetivos del proyecto.

En línea con el compromiso creciente de la comunidad científica y técnica con los retos globales, este proyecto también pretende contribuir al avance hacia los Objetivos de Desarrollo Sostenible (ODS) definidos por las Naciones Unidas. Los ODS constituyen un marco global de acción orientado a erradicar la pobreza, proteger el planeta y asegurar la

prosperidad para todos, mediante 17 objetivos concretos y 169 metas asociadas que abarcan dimensiones económicas, sociales y ambientales del desarrollo (Naciones Unidas, s.f.).

Figura 9 Objetivos de Desarrollo Sostenible (ODS) (Naciones Unidas, s.f.).

La agricultura, y en particular el sector vitivinícola, juega un papel clave en el equilibrio ambiental y en la sostenibilidad de los territorios rurales. Sin embargo, también enfrenta importantes desafíos en cuanto a su impacto ambiental y a la adopción de innovaciones tecnológicas, como la contaminación de los suelos y las aguas que puede ser producida por una aplicación en masa de productos fertilizantes y fitosanitarios. Es por esto por lo que este proyecto contribuye de manera directa a los siguientes ODS:

ODS 9: Industria, innovación e infraestructura

El proyecto impulsa la digitalización del sector agrícola mediante la integración de tecnologías avanzadas como la realidad aumentada y el aprendizaje automático. La aplicación desarrollada constituye un ejemplo tangible de cómo la innovación tecnológica puede aportar valor en el ámbito rural y contribuir a reforzar la competitividad y sostenibilidad del sector.

ODS 12: Producción y consumo responsables

El modelo de inteligencia artificial integrado en la aplicación permite detectar de forma temprana y precisa las principales enfermedades que afectan a la vid. Esta capacidad contribuye a optimizar la aplicación de tratamientos fitosanitarios, reduciendo su uso innecesario y minimizando así el impacto ambiental sobre el suelo, el agua y los ecosistemas. Además, el sistema permite a los viticultores tomar decisiones más informadas, promoviendo prácticas agrícolas más sostenibles y respetuosas con el entorno.

Con ello, el proyecto no solo persigue un avance técnico en el ámbito de la ingeniería y la inteligencia artificial, sino que también se alinea con los principios de sostenibilidad y responsabilidad ambiental, demostrando cómo las tecnologías digitales pueden ser aliadas clave en la construcción de un futuro más equilibrado y respetuoso con el medio ambiente.

3.3 ESTIMACIÓN ECONÓMICA

Para valorar económicamente el desarrollo del presente proyecto, se ha realizado una estimación detallada de los costes asociados, tanto en concepto de recursos humanos como de infraestructura y licencias. El desglose se muestra en la tabla1.

Concepto	Precio	Precio / hora	Horas	Total
Ingeniero de telecomunicaciones		14,00 €	300	4.200,00 €
Ordenador	800,00 €			800,00 €
Internet	180,00 €			180,00 €
Servidor con dominio web	120,00 €			120,00 €
Marketplace de Apple	86,00 €			86,00 €
Marketplace de Android	23,00 €			23,00 €
Unity Pro	1.110,00 €			1.110,00 €

TOTAL	**6.519,00 €**

Tabla 1 Estimación económica del proyecto.

El coste más significativo corresponde al trabajo realizado por un ingeniero de telecomunicaciones. Al tratarse el Trabajo de Fin de Máster de una asignatura de 12 créditos, esto se corresponde a 300 horas de trabajo valoradas en unos 14 €/hora, teniendo en cuenta el salario promedio de un ingeniero de telecomunicaciones junior en España de unos 27.000€ anuales (Talent.com, 2025). También es remarcable el coste de un ordenador, que dependerá del modelo, pero puede estar en torno a los 800€ (figura 10) y la conexión a internet (figura 11), unos 30€ al mes.

Figura 10 Ordenador para el proyecto (MediaMarkt, 2025).

Figura 11 Tarifa de internet (Movistar, 2025).

Además, se contemplan los costes derivados del uso de servicios en la nube de una versión más profesional (servidor para despliegue del modelo de IA). Por 3,50€, en 6 meses de desarrollo sería un coste de 21€ y vendría con todas las características de la figura 12.

Figura 12 Servidor Hostinet (Hostinet, 2025)

Hay que contemplar también las tasas obligatorias para la publicación de la aplicación en los principales *marketplaces*, con un precio de 99$ (unos 86€) para Apple (Apple, 2025) y un precio de 25$ (unos 23€) para el caso de Android (Google Play, 2019), contemplando que la aplicación se distribuya comercialmente.

Se incluyen finalmente las licencias del software de Unity Pro (figura 13) de 185€ mensuales (1.110€ en total).

Figura 13 Licencia de Unity Pro (Unity, 2025).

En conjunto, el proyecto tiene un coste total estimado de **6.519,00 €**, lo que proporciona una referencia realista sobre el valor económico que tendría un desarrollo profesional de este tipo en el mercado.

Capítulo 4. DESCRIPCIÓN DE LAS TECNOLOGÍAS

El presente capítulo describe las principales tecnologías utilizadas en el desarrollo del proyecto, que abarcan desde el entorno de programación hasta los componentes de inteligencia artificial y realidad aumentada.

4.1 APLICACIONES MÓVILES MULTIPLATAFORMA

Las aplicaciones móviles multiplataforma permiten desarrollar soluciones que pueden ejecutarse de forma nativa en los principales sistemas operativos móviles, como Android y iOS, a partir de una única base de código. Este enfoque facilita el desarrollo, reduce el esfuerzo de mantenimiento y asegura una experiencia de usuario similar en diferentes tipos de dispositivos. El desarrollo multiplataforma se apoya en herramientas y entornos que abstraen las diferencias entre plataformas, proporcionando componentes comunes y reutilizables para todos los elementos de la aplicación.

El auge de los dispositivos móviles y la necesidad de llegar a una audiencia global han impulsado la adopción de *frameworks* multiplataforma como Unity, React Native, Xamarin y Flutter. Estas tecnologías permiten compilar y desplegar aplicaciones en múltiples sistemas operativos a partir de un único código fuente, lo que acelera los ciclos de desarrollo y facilita la actualización y mantenimiento de las aplicaciones. Además, esta estrategia resulta especialmente ventajosa para proyectos que requieren presencia simultánea en varios mercados o para empresas que buscan optimizar sus recursos, tanto técnicos como humanos.

4.1.1 MOTOR DE DESARROLLO UNITY

Unity es uno de los motores de desarrollo más utilizados a nivel mundial para la creación de aplicaciones interactivas y experiencias gráficas en tiempo real, tanto en el ámbito de los videojuegos como en el desarrollo de aplicaciones móviles y de realidad aumentada. Su principal fortaleza radica en la capacidad de diseñar y desplegar aplicaciones compatibles con Android y iOS a partir de un único proyecto, lo que simplifica notablemente el proceso de desarrollo y reduce los costes de adaptación y mantenimiento.

Figura 14 Logo de Unity (Unity Technologies, s.f.-b).

Unity proporciona un entorno gráfico avanzado y un conjunto de herramientas integradas que permiten gestionar múltiples aspectos del desarrollo. Las principales herramientas son las siguientes (Unity Technologies, s.f.-a):

- **Editor de escenas y jerarquía:** Permite visualizar y editar escenas tanto en 2D como en 3D, organizar objetos y gestionar su jerarquía.
- **Ventana de proyecto:** Muestra todos los recursos (en el programa, *assets*) disponibles para el proyecto, facilitando la gestión de materiales, texturas, sonidos, scripts, etc.
- **Inspector:** Permite visualizar y editar las propiedades de cualquier objeto seleccionado en la escena.

A parte de estas herramientas genéricas, también existen herramientas específicas, que van desde la creación de interfaces de usuario, hasta la navegación y búsqueda de caminos o la integración de componentes de realidad aumentada.

Además, la plataforma cuenta con una *Asset Store*, un repositorio de recursos reutilizables que facilita la incorporación de modelos 3D, sonidos, texturas y scripts, acelerando el desarrollo y mejorando la calidad de los proyectos.

Una característica fundamental de Unity es su sistema de *scripting* basado en C#, un lenguaje moderno, orientado a objetos y ampliamente documentado, que permite programar comportamientos personalizados y controlar tanto la lógica de la aplicación como las interacciones con el usuario y el entorno virtual.

Desde el punto de vista académico, Unity es ampliamente recomendado por su flexibilidad, su comunidad activa y la abundancia de documentación y recursos de aprendizaje. Diversos estudios y proyectos universitarios destacan la idoneidad de Unity para la enseñanza y la investigación en áreas como la realidad aumentada, la visión por computador y la inteligencia artificial aplicada (Cuadros Acosta, 2020).

En resumen, Unity se consolida como una plataforma integral y versátil para el desarrollo de aplicaciones móviles multiplataforma, ofreciendo ventajas significativas en términos de productividad, portabilidad y calidad de las experiencias interactivas generadas.

4.1.2 SCRIPTS DE C#

C# es un lenguaje de programación moderno, orientado a objetos, multiplataforma, de código abierto y muy innovador. Desarrollado por Microsoft, se distingue por su robustez, facilidad de uso y flexibilidad. Su uso está muy extendido en el desarrollo de aplicaciones para Windows, pero también juega un papel crucial en el desarrollo de videojuegos, especialmente en herramientas como Unity.

Dentro de Unity, C# es el lenguaje fundamental para crear scripts que controlan tanto los aspectos de la aplicación como las interacciones dentro del entorno de desarrollo. En el contexto de aplicaciones de Realidad Aumentada, los scripts en C# tienen una relevancia significativa, ya que permiten configurar elementos clave como la cámara, las interacciones con objetos virtuales y la gestión de entradas del usuario (Holopainen, 2016). En Unity, los scripts se basan en una clase que hereda de MonoBehaviour, un componente clave que facilita la interacción con los objetos de la escena. Este enfoque no solo permite controlar elementos gráficos y visuales, como los objetos 3D, sino que también habilitas funcionalidades más avanzadas, como la comunicación con servidores externos o la gestión de datos mediante peticiones HTTP, que son esenciales para aplicaciones dinámicas y en tiempo real.

Así pues, C# no solo es el lenguaje que facilita la creación de aplicaciones, sino también el que gestiona la lógica y las interacciones que hacen posible la experiencia de Realidad Aumentada dentro de Unity.

4.2 REALIDAD AUMENTADA

La realidad aumentada (RA; en inglés, *Augmented Reality*, AR) es una tecnología que permite superponer información digital como textos, imágenes o modelos 3D sobre el entorno real, a través de dispositivos como smartphones o tabletas. A diferencia de la realidad virtual, que crea un entorno completamente inmersivo, la realidad aumentada mantiene el contexto físico y añade elementos virtuales complementarios que enriquecen la experiencia del usuario y complementan la realidad.

En el desarrollo de aplicaciones móviles, la RA integra objetos digitales, visualizaciones e información dentro del visor de la cámara del dispositivo, generando una combinación de elementos reales y virtuales. Gracias a la aparición y mejora continua de los kits de desarrollo de software (SDK) como ARCore (Google), ARKit (Apple), Vuforia y AR Foundation

(Unity), los desarrolladores pueden crear experiencias de RA sólidas y atractivas, con reconocimiento de imágenes, seguimiento de objetos y comprensión ambiental (He, 2020).

Las aplicaciones basadas en RA están experimentando una expansión notable en múltiples sectores, gracias a su capacidad para ofrecer experiencias interactivas y contextuales de alto valor añadido.

4.2.1 RA EN UNITY

Unity se ha consolidado como una de las plataformas más robustas y flexibles para el desarrollo de aplicaciones de realidad aumentada, gracias a su arquitectura modular y su potente sistema de renderizado gráfico. El paquete *AR Foundation* de Unity permite integrar fácilmente funcionalidades de RA tanto en dispositivos Android como iOS, utilizando *ARCore* y *ARKit* respectivamente (Unity Technologies, s.f.-c).

Unity facilita la gestión de la cámara del dispositivo, la superposición de la interfaz gráfica y la integración de elementos 2D y 3D sobre el flujo de vídeo en tiempo real. Además, permite el control dinámico de la interacción del usuario con los elementos de la interfaz y la gestión de la lógica de la aplicación mediante scripts en C#. Esta compatibilidad multiplataforma resulta especialmente ventajosa para proyectos que requieren desplegar soluciones RA en diferentes sistemas operativos a partir de un único proyecto de desarrollo.

4.3 INTELIGENCIA ARTIFICIAL

La inteligencia artificial (IA) es un campo de la informática que se centra en el desarrollo de sistemas capaces de realizar tareas que, tradicionalmente, requieren inteligencia humana. Entre ellas se incluyen el reconocimiento de patrones, el aprendizaje contextualizado a partir de datos, la toma de decisiones y la adaptación a entornos cambiantes (McCarthy, 2007).

En los últimos años, la IA ha experimentado un notable avance gracias al desarrollo de técnicas de aprendizaje automático (*Machine Learning*) y aprendizaje profundo (*Deep Learning*), que permiten a los sistemas mejorar su rendimiento de forma autónoma a medida que procesan más información (figura 15).

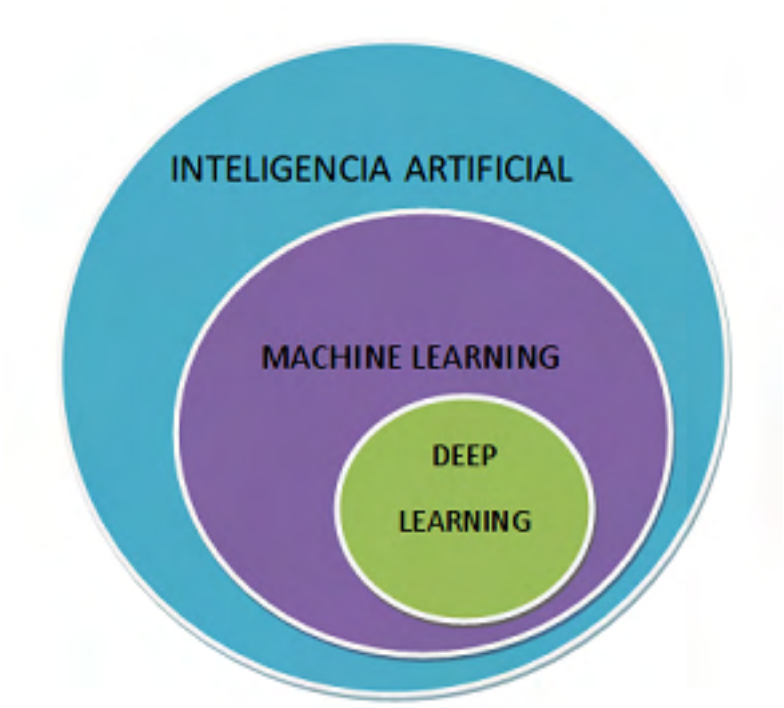


Figura 15 Inteligencia Artificial, Aprendizaje automático y Aprendizaje profundo (IAEco, s.f.).

4.3.1 APRENDIZAJE AUTOMÁTICO

El aprendizaje automático (en inglés, *Machine Learning*, ML) es una rama de la IA que se basa en el desarrollo de algoritmos capaces de aprender patrones a partir de datos. Estos algoritmos no se programan explícitamente para resolver una tarea, sino que extraen reglas y relaciones a partir de ejemplos con el fin de automatizar y optimizar la toma de decisiones (Bishop, 2006).

El proceso de aprendizaje en ML suele implicar los siguientes pasos:

1. Recopilación y preparación de un conjunto de datos de calidad.
2. Selección de un modelo adecuado.
3. Entrenamiento del modelo sobre los datos, en el que se identifican los patrones más importantes mediante una transformación matemática conocida como mapa de características, lo cual permite detectar relaciones en entornos no lineales. Esto es exclusivo del aprendizaje supervisado.
4. Validación y ajuste del modelo (también exclusivamente en el aprendizaje supervisado, ya que en el no supervisado no existen etiquetas para evaluar el modelo de la misma manera)
5. Uso del modelo para realizar predicciones o clasificaciones sobre nuevos datos.

Existen diversos tipos de aprendizaje dentro del ML, entre las que destacan el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo (figura 16). El aprendizaje supervisado utiliza datos etiquetados para que el modelo aprenda la relación entre entradas y salidas conocidas, mientras que el no supervisado busca patrones o estructuras ocultas en datos no etiquetados. Por otra parte, el aprendizaje por refuerzo se basa en la interacción del modelo con un entorno, recibiendo recompensas o penalizaciones según sus acciones, y es ampliamente utilizado en robótica y sistemas de control (Nasteski, 2017).

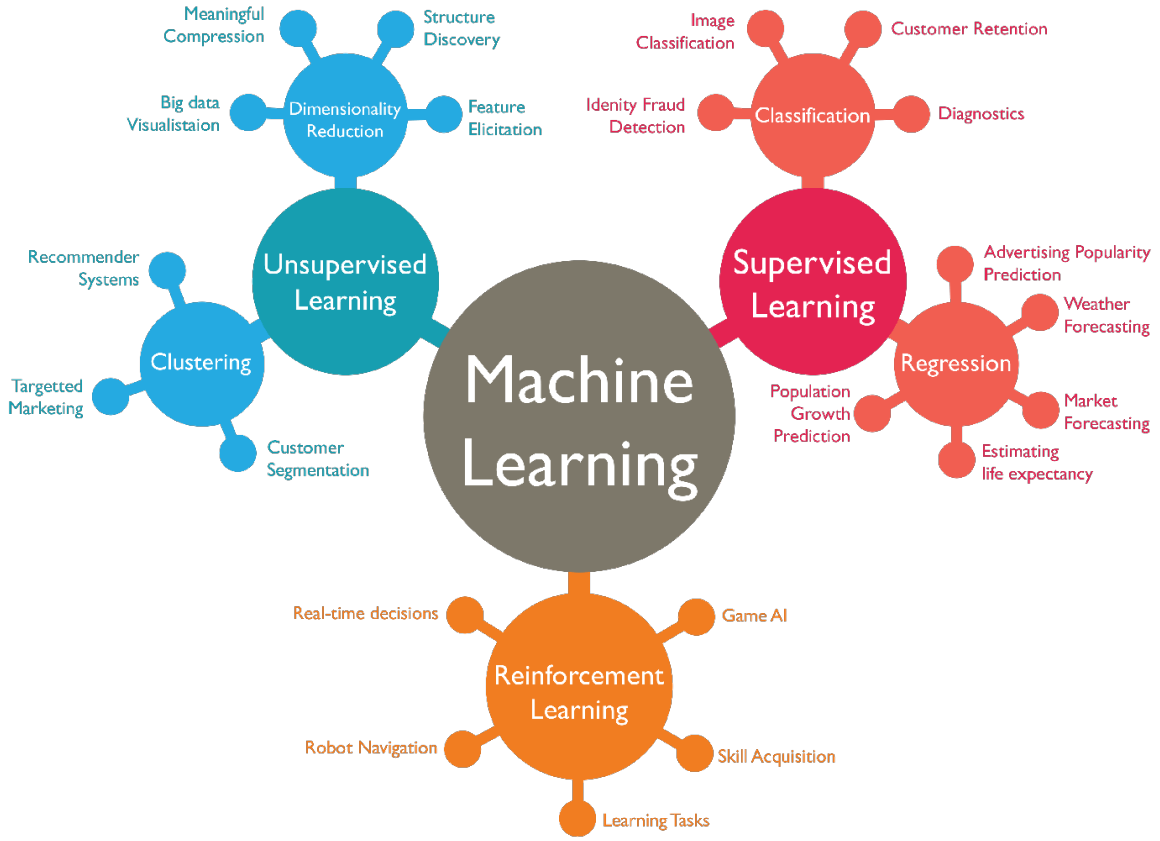


Figura 16 Clasificación del ML (Andres, 2023).

El aprendizaje automático se ha convertido en una herramienta clave para múltiples sectores, desde la agricultura hasta la salud, pasando por la industria, el comercio y los servicios financieros, permitiendo automatizar procesos, optimizar recursos y descubrir información relevante a partir de grandes volúmenes de datos.

4.3.2 APRENDIZAJE PROFUNDO

El aprendizaje profundo (en inglés, *Deep Learning*, DL) es un subcampo del aprendizaje automático que utiliza redes neuronales profundas, compuestas por múltiples capas de procesamiento, para aprender también relaciones entre datos. Las capas más cercanas a los datos de entrada aprenden características más simples mientras las más lejanas aprenden características más complejas. Estas redes han demostrado una capacidad sobresaliente para

tratar datos complejos y no estructurados, como imágenes, audio o texto, logrando resultados de vanguardia en tareas como el reconocimiento de imágenes, la traducción automática y la comprensión del lenguaje natural (Shinde & Shah, 2018).

El entrenamiento de modelos de aprendizaje profundo requiere habitualmente grandes cantidades de datos y recursos computacionales significativos, pero permite obtener sistemas con una gran capacidad de generalización y adaptación. El enfoque basado en aprendizaje profundo constituye en la actualidad uno de los pilares fundamentales de la inteligencia artificial moderna.

4.3.2.1 Redes Neuronales

Las redes neuronales artificiales son algoritmos bioinspirados que simulan el comportamiento de las neuronas del cerebro humano. El funcionamiento de una neurona artificial se basa en un proceso similar al de una neurona biológica, aunque de manera más sencilla. Las neuronas biológicas reciben señales de otras neuronas a través de sus dendritas, procesan la información en el soma, y transmiten señales eléctricas a otras neuronas a través del axón (figura 17). De manera similar, una neurona artificial recibe entradas, las procesa con una función matemática (función de activación), y produce una salida que será transmitida a las siguientes neuronas (Izaurieta & Saavedra, 2000).

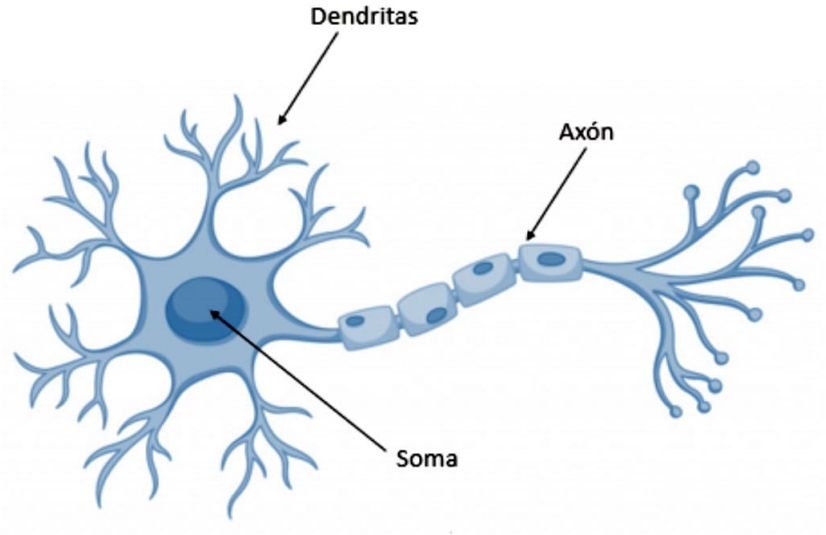


Figura 17 Neurona biológica (ABC Fichas, s.f.).

Las redes neuronales, entonces, están formadas por neuronas artificiales interconectadas, que reciben información de las neuronas anteriores y transmiten el resultado a las neuronas siguientes. Una neurona artificial es una unidad básica de procesamiento dentro de una red neuronal. Cada red neuronal tiene tres funciones principales (figura 18):

1. **Recibir entradas:** Estas entradas pueden ser cualquier tipo de dato, como valores numéricos, imágenes, o texto.
2. **Suma ponderada de las entradas:** Cada entrada recibida se multiplica por un peso asociado a la conexión entre las neuronas. Luego, todos estos productos se suman junto con un valor de sesgo (*bias*), resultando en un valor agregado que representa una combinación lineal de las entradas ponderadas.
3. **Aplicar una función de activación:** Una vez que las entradas son recibidas, la neurona realiza un cálculo para determinar si debe activarse y transmitir el resultado a la siguiente capa de neuronas.

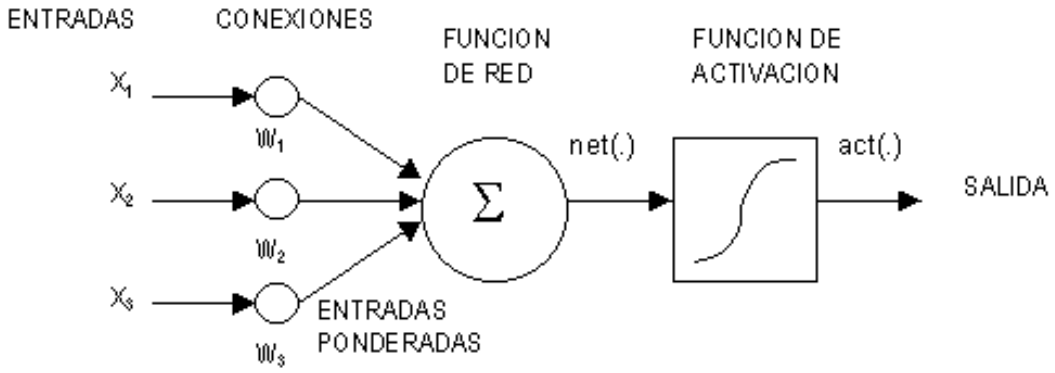


Figura 18 Funcionamiento de una red neuronal (Las Redes Neuronales, 2015).

Durante el proceso de entrenamiento de una red neuronal, los pesos son ajustados para minimizar el error en las predicciones del modelo. Este ajuste es clave para que el modelo aprenda de los datos. El proceso de ajuste de pesos es conocido como retropropagación (o *backpropagation*). En este proceso, el error en la salida de la red (la diferencia entre la predicción de la red y el valor real) se propaga de vuelta a través de la red, ajustando los pesos de cada conexión para reducir este error. Para un caso de clasificación, una función de pérdida muy común es *categorical crossentropy*, que calcula la diferencia entre las predicciones y las etiquetas de clase y que es especialmente útil cuando las etiquetas son enteros (Ramadhan et al., 2021). Los pesos se actualizan utilizando un algoritmo de optimización que ajusta los pesos para minimizar el error en el modelo. Un ejemplo de algoritmo de optimización es el optimizador Adam, (*Adaptive Moment Estimation*), que hace uso del gradiente descendiente y del *momentum* (Rojano et al., 2021). El gradiente descendiente es un algoritmo de optimización que minimiza la función de pérdida de un modelo para que, en el caso de las redes neuronales, minimizar la diferencia entre las predicciones y los valores reales. El *momentum*, a su vez, es una técnica que acelera al gradiente descendiente para evitar oscilaciones y mínimos locales y optimizar el proceso de aprendizaje.

Durante el entrenamiento, el rendimiento de la red se evalúa típicamente mediante las curvas de precisión y curvas de pérdida. La curva de precisión muestra cómo la capacidad del

modelo para clasificar correctamente los datos mejora a lo largo del entrenamiento, contabilizando los casos que se han predicho correctamente, mientras que la curva de pérdida mide el error del modelo. Estas curvas permiten monitorear el progreso y detectar problemas como el sobreajuste (*overfitting)*, que ocurre cuando se aprenden demasiado bien los detalles y se pierde capacidad de generalización, o el subajuste (*underfitting)*, el efecto contario (figura 19).

Figura 19 Comparación de un entrenamiento correcto, con sobreajuste y con subajuste (Aprende Machine Learning, 2017).

Para mejorar la capacidad de generalización de un modelo y evitar el sobreajuste, se utilizan técnicas de regularización como *dropout* y normalización. *Dropout* consiste en apagar aleatoriamente algunas neuronas durante el entrenamiento, lo que obliga a la red a no depender excesivamente de una sola neurona y mejora su capacidad para generalizar, como se puede ver en la figura 20 (Srivastava et al., 2014). Por otro lado, normalizar facilita que el proceso de entrenamiento sea más estable y rápido, lo que permite a la red aprender de manera más eficiente (Bjorck et al., 2018).

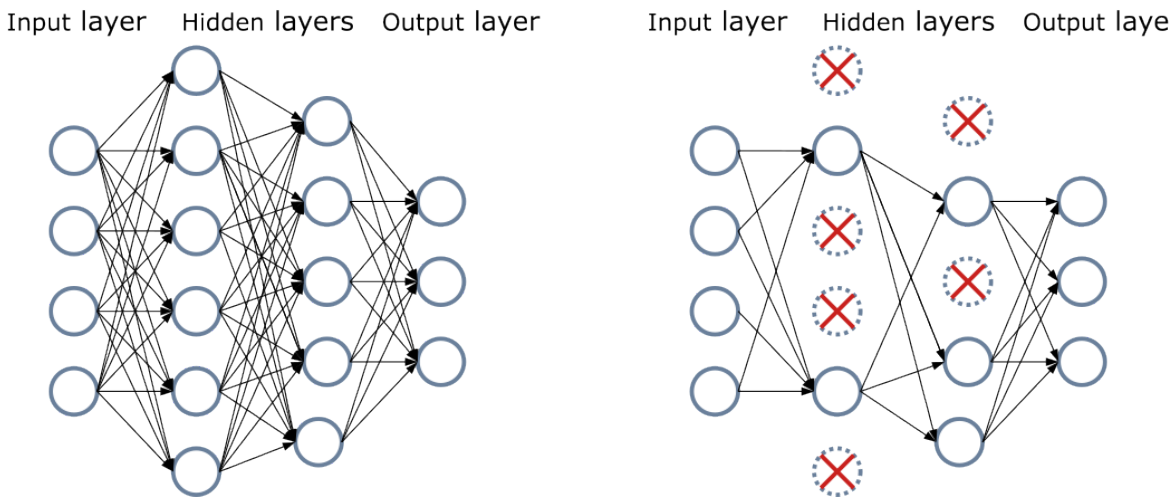


Figura 20 Ejemplo de red neuronal con dropout (Vitality Learning, 2024).

La red más sencilla que existe es el perceptrón (figura 21). Se trata de uno de los primeros modelos de neuronas artificiales y es la base de las redes neuronales más complejas. Tiene una única capa, y las neuronas tienen una función activación de tipo escalón para tomar decisiones binarias. El proceso más básico que sigue un perceptrón es el de recibir entradas multiplicadas por un peso, hacer una suma ponderada de las entradas con los pesos, pasar por una función de activación (como la función escalón descrita en la figura 22) para determinar si debe activarse o no, y emitir una señal de salida.

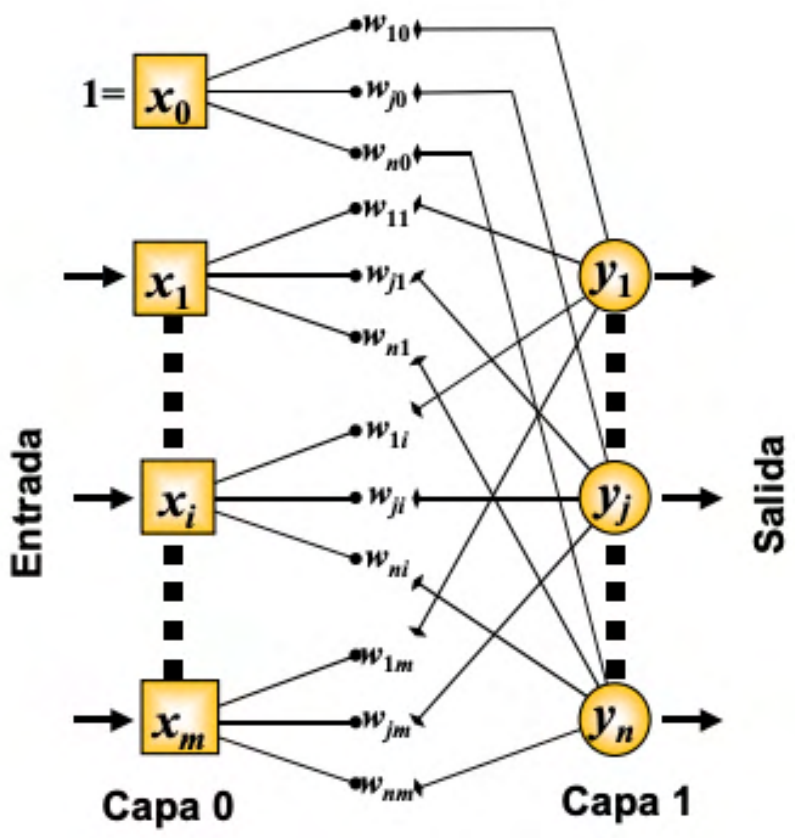


Figura 21 Ejemplo de red neuronal unicapa (Izaurieta & Saavedra, 2000).

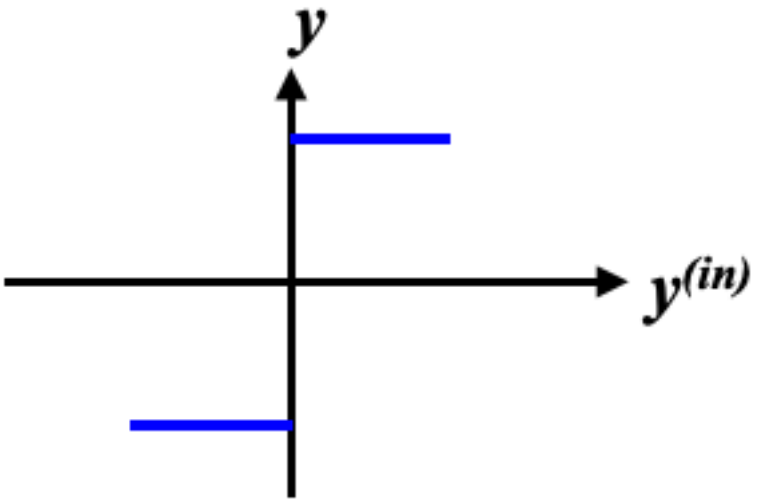


Figura 22 Función de activación escalón (Izaurieta & Saavedra, 2000).

Un ejemplo de red neuronal son las redes neuronales convolucionales, que tienen una gran relevancia para este proyecto al ser la base del modelo predictivo para detectar enfermedades de la vid.

4.3.2.2 Redes Neuronales Convolucionales

Las redes neuronales convolucionales (en inglés, *Convolutional Neural Networks*, CNN) son una arquitectura especializada en procesar datos en forma de múltiples vectores como pueden ser las imágenes (por ejemplo, una imagen a color son 3 vectores en 2D, para los canales rojo, verde y azul) y forman parte de las redes neuronales profundas (LeCun et al., 2015). Las CNN son especialmente relevantes por su capacidad de capturar patrones locales mediante filtros convolucionales. Estos filtros son matrices de pequeños valores (generalmente 3x3 o 5x5) que se deslizan sobre la imagen de entrada a través de un proceso llamado convolución.

El filtro se mueve de izquierda a derecha y de arriba hacia abajo sobre la imagen (o mapa de activación de la capa anterior), realizando una multiplicación punto a punto entre los valores del filtro y los valores de la imagen en cada posición del filtro (figura 23). Esta operación da como resultado un valor que se coloca en un nuevo mapa de características, que representa los valores aprendidos por el filtro. Estos patrones pueden ser simples, como bordes o colores si se trata de imágenes, o más complejos, como texturas y formas, dependiendo de la profundidad de la red y de los filtros utilizados.

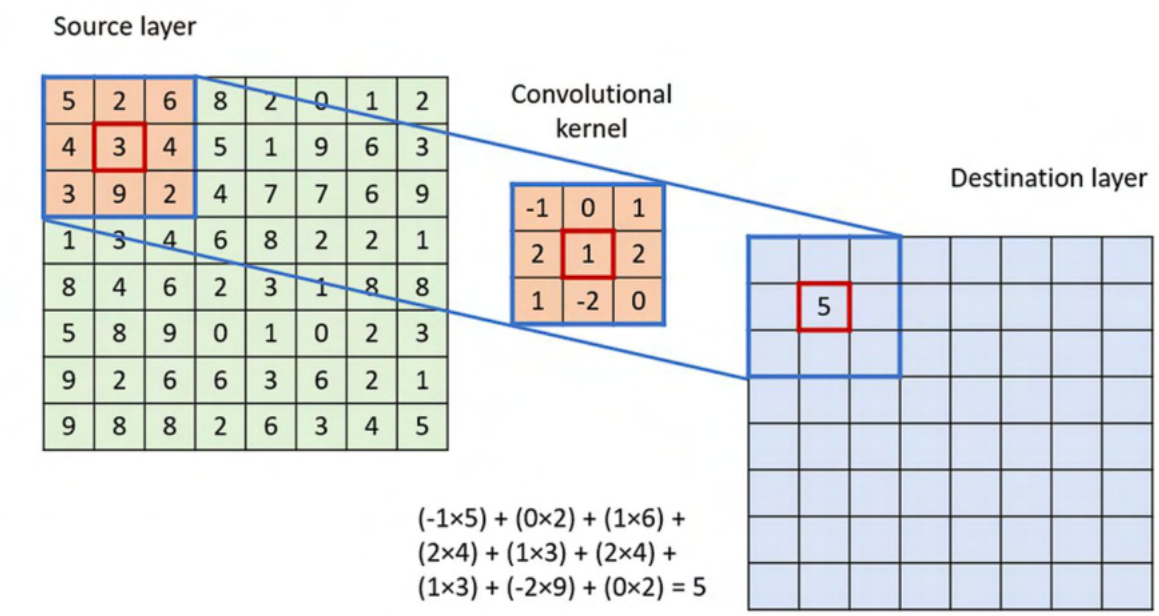


Figura 23 Ejemplo de funcionamiento de un filtro convolucional (GeeksforGeeks, 2025a).

Cada filtro está diseñado para detectar un patrón específico, y a medida que los filtros se aplican a las distintas capas de la red, los patrones que representan se vuelven más complejos y abstractos, lo que permite a la red aprender representaciones jerárquicas de la imagen.

Una CNN está formada por diferentes capas, cada una con una función específica. Para mayor claridad, se explican con el ejemplo de la figura 24.

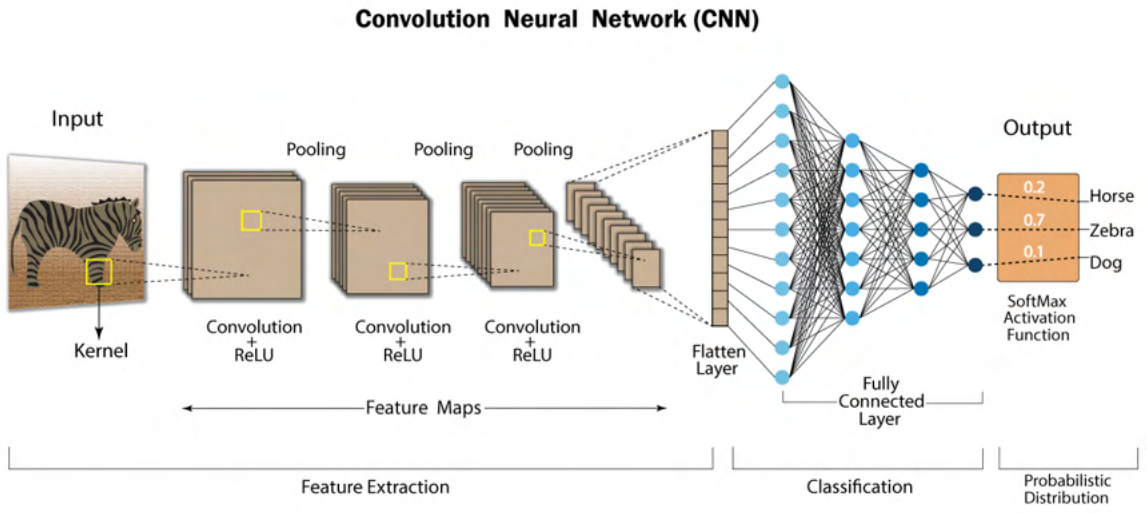


Figura 24 Capas de una CNN (Gutiérrez, 2023).

Como se observa en la figura 24, un ejemplo práctico es la clasificación de una imagen según la especie de animal representada. En este caso, se utiliza una imagen de una cebra. Primero, se aplica una matriz (*kernel*) que se desplaza a través de la imagen, realizando la operación de convolución en cada una de las capas. Además, se aplica una función de activación ReLU (figura 25), que, al introducir no linealidades, permite a la red aprender patrones más complejos. Aunque ReLU es una de las funciones de activación más populares, existen otras alternativas, como la sigmoide y la tangente hiperbólica, que también se utilizan en redes neuronales para diversas aplicaciones.

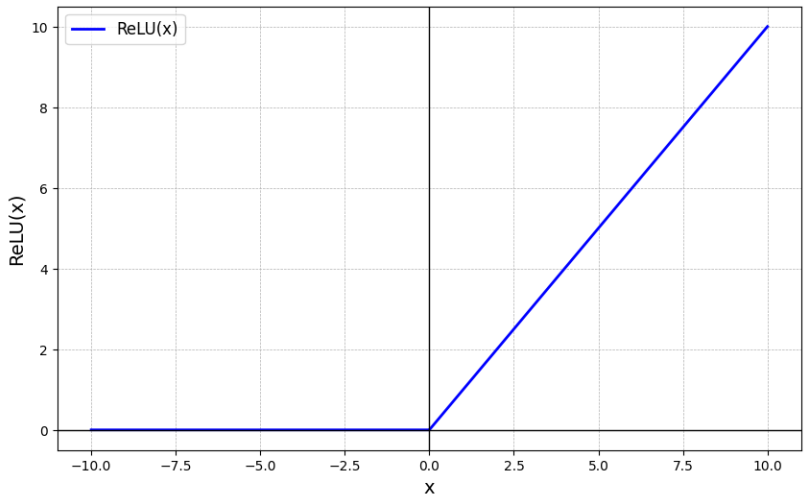


Figura 25 Función de activación ReLU (GeeksforGeeks, 2025b).

El proceso de *pooling* reduce progresivamente la dimensión de las características extraídas. Posteriormente, la capa de *flatten* convierte los mapas de características en un vector adecuado para la clasificación final. En la capa completamente conectada (también conocida como capa densa), donde cada nodo está vinculado a todos los nodos de la capa anterior, se toman las decisiones de clasificación basándose en las características extraídas de la imagen. Finalmente, la capa de salida utiliza una función de activación *softmax* (aunque podría ser otra) que convierte los valores de salida de la red en probabilidades, y se excita la neurona con mayor probabilidad, que acaba siendo la clase predicha.

Finalmente, en el ámbito del aprendizaje profundo es habitual emplear técnicas complementarias que mejoran el rendimiento de las redes neuronales convolucionales. Entre ellas destacan el aumento de datos (*data augmentation*), que genera nuevas imágenes a partir de transformaciones sobre los datos originales para aumentar la diversidad del conjunto de entrenamiento, y el uso de redes preentrenadas, que permiten reutilizar modelos ya entrenados sobre grandes bases de datos para adaptarlos a nuevas tareas con menor esfuerzo computacional (Goodfellow et al., 2016).

4.4 MÉTODO POST

Dentro del protocolo HTTP (*HyperText Transfer Protocol*), el método POST es una herramienta esencial para enviar datos a servidores para que sean procesados y recibir respuestas que pueden ser utilizadas directamente en la aplicación del cliente (Mozilla Contributors, 2024). Se trata de una de las maneras de indicar la acción que se desea realizar para un recurso en concreto, junto a los métodos GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE y PATCH. La principal diferencia del método POST con los anteriores es que se usa para enviar una entidad a un recurso causando un cambio en el estado del servidor. Algunos de los beneficios de usar el método POST son:

- **Escalabilidad:** el método POST no tiene límite en la cantidad de datos enviados, por lo que es ideal para manejar archivos pesados como imágenes.
- **Flexibilidad:** permite el envío de una gran variedad de datos, ya sean texto, imágenes o datos binarios.

4.5 XCODE

Xcode es el entorno de desarrollo integrado oficial de Apple y sirve para crear todo tipo de aplicaciones destinadas a sus sistemas operativos. Esta herramienta permite compilar, ejecutar, depurar y distribuir aplicaciones en dispositivos Apple (Apple Inc., 2025).

Figura 26 Logo Xcode (Apple, s.f.).

Xcode cuenta con varias herramientas para el desarrollo de aplicaciones del ecosistema Apple, como un editor de código avanzado, un compilador, un diseñador de interfaces gráficas, un simulador de dispositivos, herramientas de análisis de rendimiento y gestión de versiones con Git.

En este caso, permite gestionar aspectos clave como las firmas de desarrollo, la configuración de un identificador único para cada aplicación, la selección del dispositivo destino deseado o incluso la aplicación en la App Store.

4.6 SERVIDOR WEB

Un servidor web es una tecnología fundamental dentro de las arquitecturas cliente-servidor modernas, encargada de gestionar, procesar y responder a las solicitudes enviadas por los clientes, que generalmente suelen ser aplicaciones o navegadores, mediante protocolos como HTTP o HTTPS. Su función principal es servir contenidos estáticos, como pueden ser HTML, CSS o imágenes o contenidos dinámicos mediante el procesamiento de scripts en el lado servidor a través de internet o redes locales (Mozilla Contributors, 2025).

Desde un punto de vista técnico, un servidor web recibe peticiones de los clientes, las interpreta y envía una respuesta adecuada, que puede consistir en una página web, un archivo

multimedia o una respuesta generada por una aplicación *backend*. Para ello, se utilizan tanto servidores genéricos como Apache o NGINX, como soluciones más ligeras y específicas embebidas en *frameworks*, como Flask o Express.js.

Además de servir contenido, los servidores web pueden incorporar múltiples funcionalidades como:

- Gestión de sesiones y cookies para usuarios autenticados.
- Procesamiento de formularios y APIs REST mediante métodos HTTP como GET o POST.
- Integración con bases de datos y otros servicios a través de middlewares o controladores.
- Cifrado de comunicaciones mediante HTTPS mediante certificados digitales.

En el desarrollo de aplicaciones móviles modernas, los servidores web actúan como intermediarios esenciales entre el cliente y el *backend*, permitiendo la consulta, envío o actualización de datos de forma asíncrona. Su correcta configuración y escalabilidad son aspectos clave en el rendimiento, la seguridad y la experiencia del usuario final.

Capítulo 5. METODOLOGÍA

La metodología seguida en este proyecto combina un enfoque modular y práctico basado en el desarrollo iterativo. El sistema completo se estructura en varios bloques tecnológicos interdependientes, cada uno con una función específica dentro del flujo general de la aplicación. A continuación, se detallan los principales componentes y su rol en el proceso de desarrollo:

- **Unity: Aplicación móvil con *AR Foundation***

  Se hace uso del motor Unity como entorno principal de desarrollo de la aplicación móvil. Gracias a *AR Foundation*, se pudo construir una experiencia de realidad aumentada multiplataforma compatible con *ARKit* (iOS) y *ARCore* (Android), permitiendo superponer información específica directamente sobre el entorno físico del usuario.

- **Información geolocalizada de la vid**

  Utilizando tecnología GNSS, se identifican las distintas parcelas dentro del viñedo y se muestra información relevante de cada una de ellas, desde las tareas pendientes hasta los rendimientos.

- **Base de datos**

  Se hace uso de una base de datos pública compuesta por imágenes de hojas de vid en diferentes estados sanitarios (sanas y con 3 tipos de enfermedades distintas). Este dataset constituye la fuente de datos necesaria para entrenar el modelo de aprendizaje automático.

- **Red neuronal convolucional**

Mediante bibliotecas como *TensorFlow* y *Keras*, se diseña y entrena un modelo de inteligencia artificial capaz de clasificar automáticamente enfermedades de la vid. Este modelo produce como salida predicciones rápidas y precisas al facilitarle la imagen de una hoja de vid.

- **Servidor web**

  Se hace uso de un servidor web ligero usando *Flask* para alojar el modelo de IA y permitir su consulta desde la aplicación móvil mediante peticiones POST. Este servidor se encarga de recibir las imágenes, procesarlas y devolver el diagnóstico en tiempo real.

- **Integración**

  Mediante scripts en C# dentro de Unity se controla el flujo de la aplicación, desde gestionar las peticiones al servidor, el envío de imágenes y la recepción de resultados. Esta capa de integración asegura la conexión fluida entre la interfaz móvil, el servidor y el modelo de IA.

- **Diseño**

  Un foco especialmente relevante es el de diseñar una interfaz clara, accesible y visualmente atractiva, adaptada al uso en la parcela agraria. El diseño prioriza la simplicidad de uso, la legibilidad y la eficiencia en la interacción del usuario con la app.

Este enfoque metodológico modular permite desarrollar y probar cada componente de manera independiente antes de su integración final, favoreciendo así la escalabilidad del proyecto y la posible incorporación futura de nuevas funcionalidades.

En el siguiente capítulo se mostrará los pasos precisos y la configuración y despliegue de cada una de las partes vistas en la metodología.

Capítulo 6. SISTEMA DESARROLLADO

Este capítulo describe en detalle el sistema desarrollado como resultado del presente proyecto. El sistema se compone de una aplicación móvil multiplataforma desarrollada en Unity con el paquete *AR Foundation*, que permite superponer información contextual sobre el entorno real mediante la cámara del dispositivo. Concretamente, gracias a la geolocalización, se permite mostrar información sobre las tareas pendientes y los rendimientos de la parcela identificada. Además, la aplicación está conectada a un servidor web que aloja el modelo de inteligencia artificial previamente entrenado, encargado de realizar el diagnóstico sanitario de las hojas de vid a partir de imágenes capturadas en una parcela. La figura 27 ilustra esquemáticamente este sistema.

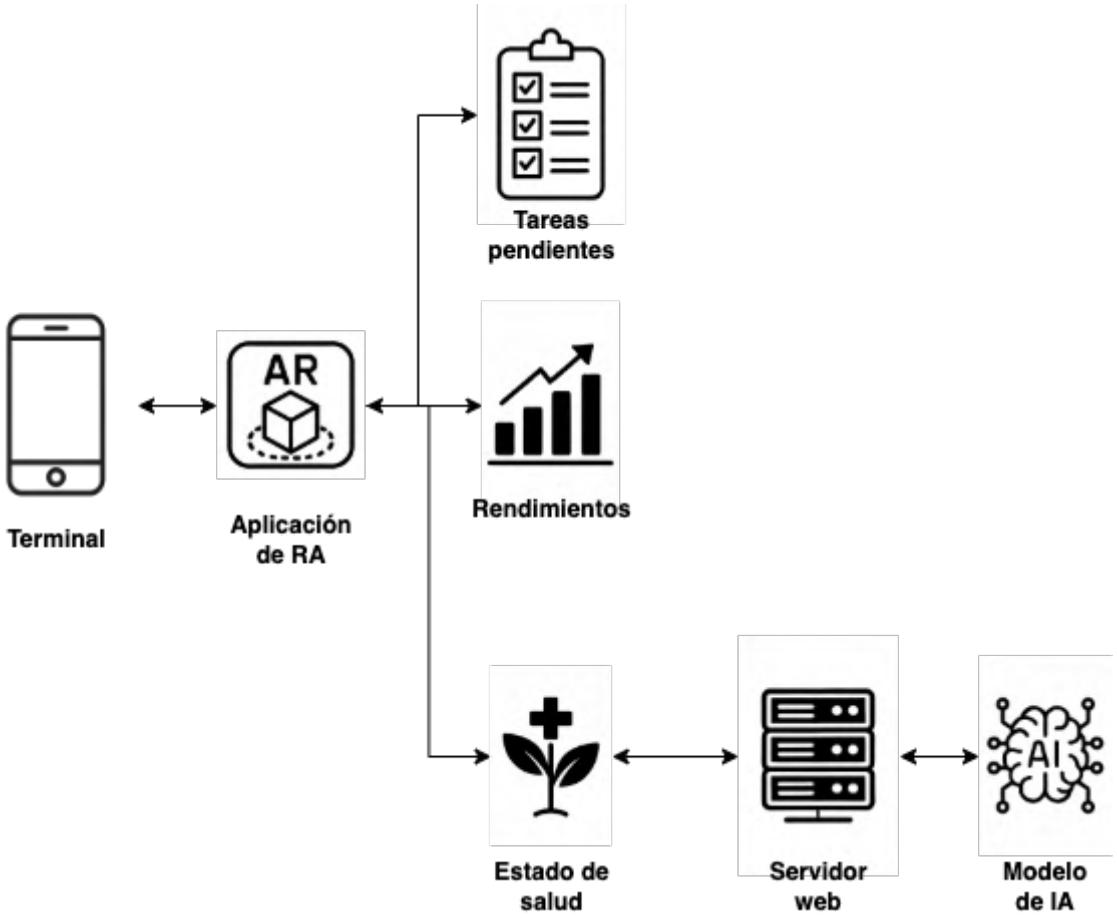


Figura 27 Diagrama del sistema desarrollado.

A lo largo de este capítulo se describen los elementos que conforman la aplicación móvil, la estructura de la interfaz de usuario, los componentes de realidad aumentada integrados, y el flujo completo de funcionamiento del sistema, desde la interacción del usuario hasta la obtención de los resultados de diagnóstico.

6.1 APLICACIÓN UNITY

La aplicación móvil desarrollada en este proyecto ha sido implementada utilizando Unity, un motor multiplataforma ampliamente utilizado para la creación de entornos interactivos en tiempo real. En concreto, se ha utilizado Unity con el paquete *AR Foundation*, lo que

permite integrar de forma sencilla las funcionalidades de realidad aumentada tanto en dispositivos Android como iOS, a través de *ARCore* y *ARKit* respectivamente.

El diseño se ha llevado a cabo dentro de una única escena denominada *SampleScene*, que organiza todos los elementos visuales e interactivos del sistema. Esta escena se divide funcionalmente en tres bloques principales: elementos de realidad aumentada, interfaz de usuario (UI) y componentes de gestión de eventos y cámara.

6.1.1 ELEMENTOS DE REALIDAD AUMENTADA

Para gestionar la experiencia RA, se ha incorporado una estructura estándar de *AR Foundation*, formada por los componentes de la figura 28:

***AR Session*:** componente encargado de gestionar la sesión de realidad aumentada, la activación de los sensores del dispositivo y el seguimiento espacial.

***XR Origin (AR Rig)*:** objeto principal que contiene la cámara (*Main Camera*) y otros elementos asociados al posicionamiento y seguimiento del usuario en el entorno RA. Aunque existe un objeto *Object Spawner*, este forma parte del contenido de la plantilla base y no se ha utilizado en la versión final de la aplicación.

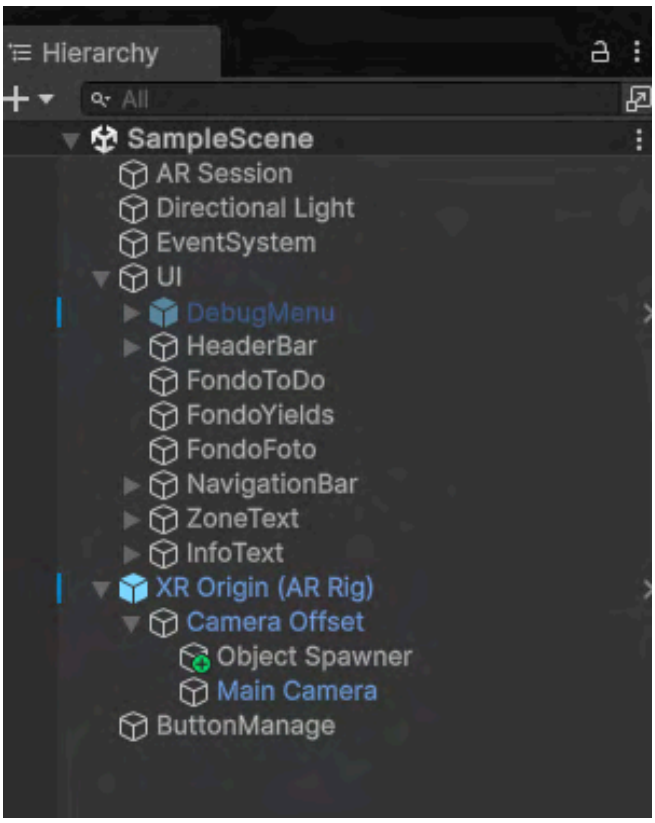


Figura 28 Jerarquía de la aplicación de Unity.

6.1.2 INTERFAZ DE USUARIO (UI)

La capa de interfaz de usuario se ha organizado bajo el objeto UI y está compuesta por distintos elementos gráficos y funcionales que permiten al usuario interactuar con la aplicación (figura 29). Entre ellos destacan:

***HeaderBar*:** barra superior de color verde oscuro que contiene el título de la aplicación (*VineGuard*), lo que aporta una identidad visual clara.

***ZoneText*:** campo de texto que muestra el número de la parcela en la que se encuentra el usuario. Este dato se actualiza automáticamente a partir del posicionamiento GNSS.

***InfoText*:** zona de información central utilizada para mostrar mensajes contextuales. En particular, se emplea para mostrar el resultado del diagnóstico recibido desde el servidor tras

el envío de una imagen y para mostrar los rendimientos y tareas de la parcela en la que se encuentra el usuario.

Figura 29 Interfaz de usuario implementado.

***NavigationBar*:** barra inferior que contiene los tres botones principales de la aplicación, representados mediante iconos intuitivos (figura 30):

- Botón de tareas pendientes (icono de lista con *check*).
- Botón de rendimientos (icono de gráfico de columnas con flecha ascendiente).
- Botón de captura de imagen (icono de cámara).

Figura 30 Botones de la aplicación.

Para asegurar la correcta visibilidad de estos botones sobre la imagen en vivo capturada por la cámara, se han incorporado fondos de color ligeramente opaco en cada uno (*FondoToDo*, *FondoYields* y *FondoFoto*), cuya única finalidad es proporcionar contraste visual y mejorar la legibilidad de los elementos superpuestos.

6.1.3 GESTIÓN DE EVENTOS DE INTERFAZ

La escena incluye un objeto *EventSystem*, necesario para gestionar correctamente la interacción con elementos de UI en Unity. Asimismo, se ha incorporado un objeto *ButtonManage*, que actúa como contenedor de lógica para los botones, permitiendo alternar entre los distintos paneles informativos. La lógica interna de estos botones y el control de flujo de datos se abordará en detalle en el apartado 6.5.

Este diseño modular y claro permite ofrecer una experiencia de usuario agradable, donde la visibilidad, simplicidad y respuesta inmediata son factores clave. La organización en bloques y el uso de elementos visuales bien contrastados favorecen la comprensión intuitiva del sistema y su futura ampliación.

6.2 BASE DE DATOS

Dado el objetivo de entrenar un modelo de inteligencia artificial capaz de detectar enfermedades en la vid, resulta esencial contar con una base de datos adecuada que permita alimentar y ajustar el modelo durante la fase de entrenamiento.

Para ello, se ha recurrido a Kaggle, una plataforma web líder en ciencia de datos y aprendizaje automático, que reúne a una extensa comunidad de profesionales, estudiantes e investigadores. Kaggle facilita la colaboración abierta, la búsqueda y publicación de conjuntos de datos sobre una gran diversidad de temáticas, y lo hace de forma gratuita. Por este motivo, se ha utilizado esta plataforma como fuente principal para la obtención del dataset (Banachewicz & Massaron, 2022).

A la hora de escoger, dado que el objetivo principal de este trabajo no es el de desarrollar un diagnóstico exhaustivo de la salud de una vid, sino integrar varios elementos de gestión y diagnóstico básico, se ha priorizado el uso de una base de datos con un número reducido de clases, para poder determinar los 3 tipos más comunes de enfermedades o si la planta está sana.

La base de datos usada tiene como título "*Augmented Grapevine Disease Dataset*", y hace uso de otra base de datos de Kaggle llamada "*grape_dataset*". Básicamente, la base de datos final consta de las imágenes de esta base de datos (1656 imágenes por clase) y se augmentan hasta las 3000 imágenes por clase, mediante aumento de tamaño (*zoom*), volteo horizontal, volteo vertical, rotación, modificación del brillo y deformación por cizalla (*shearing*). Las imágenes elegidas tienen unas dimensiones de 224 x 224 píxeles y son a color.

Esta base de datos, entonces, contiene 4 clases con 3000 imágenes por clase, es decir, 12000 imágenes en total. Las 4 clases que la conforman son las siguientes: "*Black Rot*", "*ESCA*", "*Healthy*" y "*Leaf Blight*", cuya traducción en castellano es podredumbre negra, enfermedad de la madera, saludable y quemadura foliar, y se trata de algunas de las enfermedades más típicas en la vid, reconocibles por el estado de sus hojas.

La **podredumbre negra** es causada por el hongo *Guignardia bidwellii*, que se propaga principalmente en climas cálidos y húmedos (figura 31). En las hojas se manifiesta como una serie de manchas marrones con bordes oscuros, generalmente circulares. Puede causar pérdidas severas si no se detecta a tiempo, especialmente en los racimos jóvenes (Lixandru & Fendrihan, 2023).

La **enfermedad de la madera**, en cambio, es causada por un complejo de hongos xilófagos, entre los que destacan la *Phaeomoniella chlamydospora*, la *Phaeoacremonium minimum* y la *Fomitiporia mediterranea*. En las hojas se manifiesta como manchas cloróticas y necróticas en forma de alas de mariposa entre los nervios (figura 32), y se transmite a través de heridas de poda, especialmente en plantas adultas. Se trata de una enfermedad degenerativa que no tiene una cura definitiva, que reduce la productividad y puede acabar matando a la planta progresivamente (Graniti, s.f.).

Por otro lado, la **quemadura foliar** puede ser causada por patógenos o también por exceso de radiación solar o viento caliente. Se manifiesta como manchas necróticas marrones o negras de forma irregular que suelen aparecer primero en los márgenes de las hojas (figura 33). Puede reducir la fotosíntesis y hasta afectar la calidad del fruto si es extensa (Rajarshi, s.f.).

Finalmente, una hoja en estado **sano** se caracteriza por tener un color verde homogéneo, sin manchas, necrosis ni deformaciones. Un ejemplo de ello es la hoja de la figura 34.

Figura 31 Hoja infectada por podredumbre negra.

Figura 32 Hoja infectada por enfermedad de la madera.

Figura 33 Hoja infectada por la quemadura foliar.

Figura 34 Hoja sana.

Así pues, con esta base de datos se dispone de 3000 ejemplos de cada uno de estos estados de la vid que se pueden usar para crear un modelo de IA predictivo.

6.3 MODELO DE IA

En este apartado se define el proceso para conseguir un modelo de Inteligencia Artificial predictivo capaz de clasificar a una planta dentro de una de las 4 clases de la base de datos explicada en el apartado anterior.
Para entrenar dicho modelo, se hace uso de las redes neuronales convolucionales. Como se ha explicado anteriormente en la descripción de tecnologías, las redes neuronales convolucionales son un tipo de red neuronal profunda especialmente potente para el procesamiento de imágenes, como en este caso.

Para entrenar la CNN deseada para predecir enfermedades, se plantea el siguiente flujo:

1. Importación de librerías (Anexo 1)
2. Carga y preparación del conjunto de datos
3. Preprocesado de las imágenes y aumento de datos
4. Optimización del flujo de datos
5. Construcción del modelo CNN
6. Entrenamiento del modelo
7. Evaluación de resultados

Todos estos pasos se ejecutan dentro de un *notebook* de Python.

6.3.1 CARGA Y PREPARACIÓN DEL CONJUNTO DE DATOS

El dataset obtenido de Kaggle con 12.000 imágenes es la base de esta red convolucional. En este caso, el directorio raíz se denomina *Final Training Data* y se estructura en 4 carpetas,

una por cada clase, y cada una contiene todas las imágenes de dicha clase. Esta estructura permite a TensorFlow asignar automáticamente a cada imagen su correspondiente etiqueta de clase, lo que facilita su carga y preprocesamiento.

En el notebook se define la siguiente función para cargar y dividir automáticamente el conjunto de datos:

```
# Configuración inicial
IMG_SIZE = 224
BATCH_SIZE = 32
DATA_DIR = 'Final Training Data'

# Carga del dataset desde carpetas
def load_and_preprocess_data(data_dir):
    dataset = tf.keras.utils.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="training",
        seed=123,
        image_size=(IMG_SIZE, IMG_SIZE),
        batch_size=BATCH_SIZE,
        label_mode='categorical'
    )

    validation_dataset = tf.keras.utils.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="validation",
        seed=123,
        image_size=(IMG_SIZE, IMG_SIZE),
        batch_size=BATCH_SIZE,
        label_mode='categorical'
    )

    return dataset, validation_dataset

# Llamada a la función
train_dataset, val_dataset = load_and_preprocess_data(DATA_DIR)
```

Esta función permite cargar las imágenes desde el directorio, redimensionarlas a un tamaño uniforme de 224x224 píxeles y agruparlas en lotes (*batches*) de 32 imágenes. Además, realiza automáticamente una división del conjunto en dos subconjuntos: el 80 % de las imágenes se emplean para entrenamiento y el 20 % restante para validación, utilizando una semilla aleatoria fija (*seed*=123). Al fijar esta semilla de aleatoriedad, se asegura que la

partición entre los subconjuntos de entrenamiento y test sea siempre la misma en cada ejecución, permitiendo así comparar de forma coherente los resultados obtenidos en diferentes entrenamientos.

La opción *label_mode='categorical'* convierte las etiquetas en codificación *one-hot*, lo cual es necesario para problemas de clasificación multiclase. El modo *one-hot* convierte cada etiqueta en un vector binario donde solo una posición, la de la clase correcta, vale 1 y el resto son ceros.

Este paso es esencial para asegurar que el modelo aprende de un conjunto de datos suficientemente grande y se evalúa con ejemplos que no ha visto previamente, lo cual permite medir su capacidad de generalización y reducir el riesgo de sobreajuste.

6.3.2 PREPROCESADO DE LAS IMÁGENES Y AUMENTO DE DATOS

En el modelo, todas las imágenes se normalizan de modo que todos los valores queden entre 0 y 1. Esta normalización es importante para estabilizar el proceso de entrenamiento, ya que la red neuronal trabaja mejor con valores en rangos reducidos y homogéneos.

```
layers.Rescaling(1./255.0, input_shape=(IMG_SIZE, IMG_SIZE, 3))
```

Para mejorar la capacidad de generalización del modelo y evitar el sobreajuste, se aplica una serie de transformaciones aleatorias a las imágenes del conjunto de entrenamiento, conocidas como aumento de datos. Esto permite que el modelo vea versiones ligeramente distintas de las mismas imágenes, como si fueran nuevos ejemplos. Estas transformaciones incluyen giros, zoom, cambios de contraste y volteos, todas de forma aleatoria.

```
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
    layers.RandomContrast(0.2),
])
```

Posteriormente, estas transformaciones se aplican solo al conjunto de entrenamiento, y no al de validación, para que el modelo se evalúe de forma justa con imágenes sin alterar.

```
def augment_dataset(dataset):
    return dataset.map(
        lambda x, y: (data_augmentation(x, training=True), y),
        num_parallel_calls=tf.data.AUTOTUNE
    )

train_dataset = augment_dataset(train_dataset)
```

6.3.3 OPTIMIZACIÓN DEL FLUJO DE DATOS

Una vez cargado, preprocesado y aumentado el conjunto de datos, se optimiza el flujo de datos para mejorar la eficiencia durante el entrenamiento. Esto es muy útil cuando se trabaja con grandes volúmenes de imágenes, ya que permite reducir el tiempo de entrenamiento. En *TensorFlow*, esta optimización se realiza aplicando tres operaciones clave sobre el conjunto de datos.

La primera operación es *cache()*, que almacena los datos en memoria tras la primera lectura, de forma que no se tenga que volver a cargar y procesar desde el disco en cada época. Por otro lado, se hace *shuffle(buffer_size)*, que mezcla aleatoriamente las imágenes antes de cada época. Esto evita que el modelo aprenda patrones por el orden en que se presentan los datos. Finalmente, la operación *prefetch(buffer_size=tf.data.AUTOTUNE)* permite preparar el siguiente lote de datos mientras el modelo está entrenando con el lote actual. Gracias a esto, se eliminan los tiempos de espera entre lotes, aprovechando mejor la CPU disponible. Estas transformaciones se aplican tanto al conjunto de entrenamiento como al de validación, excepto el *shuffle*, que solo se aplica al entrenamiento.

```
AUTOTUNE = tf.data.AUTOTUNE

train_dataset =
train_dataset.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_dataset = val_dataset.cache().prefetch(buffer_size=AUTOTUNE)
```

6.3.4 CONSTRUCCIÓN DEL MODELO CNN

La arquitectura del modelo se basa en una combinación de una red preentrenada y capas personalizadas, lo que permite aprovechar el aprendizaje previo en grandes conjuntos de datos y adaptarlo a la tarea específica de clasificación de enfermedades de la vid. Concretamente, se utiliza MobileNetV2 para extraer las características. MobileNetV2 es una arquitectura de red neuronal convolucional desarrollada por Google, optimizada para dispositivos móviles y entornos con recursos limitados. Este modelo es ligero, rápido y eficaz, y ya ha sido entrenado previamente en millones de imágenes. Se carga sin su capa de salida y no se entrena de nuevo (para preservar el conocimiento aprendido).

```
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False,
    weights='imagenet'
)
base_model.trainable = False
```

Sobre la salida de MobileNetV2 se añaden varias capas para perfeccionar el entrenamiento. Concretamente, se añade una capa de normalización, una de *pooling* que reduce el mapa de características, una de *dropout* para aprender combinaciones no lineales de las características extraídas y se usa *softmax* para obtener las probabilidades de pertenecer a cada clase.

```
model = keras.Sequential([
    layers.Rescaling(1./255.0, input_shape=(IMG_SIZE, IMG_SIZE, 3)),
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.3),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(NUM_CLASSES, activation='softmax')
```

Esta combinación permite un entrenamiento eficiente y efectivo gracias al conocimiento ya integrado en el modelo base.

6.3.5 ENTRENAMIENTO DEL MODELO

El modelo de red neuronal convolucional se ha entrenado durante 10 épocas, utilizando el conjunto de entrenamiento previamente definido y preprocesado. En cada época, el modelo recorre todas las imágenes del conjunto de entrenamiento y ajusta los pesos de sus capas para minimizar el error de predicción.

Para ello, se ha utilizado el algoritmo de optimización Adam, y la función de pérdida seleccionada ha sido *categorical_crossentropy*, muy popular en problemas de clasificación multiclase con etiquetas codificadas en formato *one-hot*. La métrica principal de evaluación empleada ha sido la precisión, tanto sobre el conjunto de entrenamiento como sobre el conjunto de validación.

```
history = model.fit(
    train_dataset,
    epochs=EPOCHS,
    validation_data=val_dataset,
    verbose=1
)
```

6.3.6 EVALUACIÓN DE RESULTADOS

Tras el entrenamiento del modelo, se ha procedido a evaluar su rendimiento mediante el análisis de la evolución de la precisión y de la función de pérdida, tanto sobre el conjunto de entrenamiento como en el de validación.

El objeto *history* generado por *Keras* durante el entrenamiento proporciona esta información, permitiendo visualizar de forma clara cómo ha evolucionado el aprendizaje de la red, mediante las curvas de precisión y pérdida.

```
import matplotlib.pyplot as plt

# Evolución de la precisión
plt.plot(history.history['accuracy'], label='Precisión en entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión en
validación')
plt.title('Precisión del modelo')
```

```
plt.ylabel('Precisión')
plt.xlabel('Época')
plt.legend()
plt.show()

# Evolución de la pérdida
plt.plot(history.history['loss'], label='Pérdida en entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida en validación')
plt.title('Pérdida del modelo')
plt.ylabel('Pérdida')
plt.xlabel('Época')
plt.legend()
plt.show()
```

A partir del análisis de ambas gráficas, se observa que el modelo alcanza una precisión muy elevada tanto en entrenamiento como en validación (figura 35), lo que indica un buen rendimiento. Las curvas de pérdida (figura 36) descienden de forma progresiva, lo que confirma que el aprendizaje ha sido estable. Tampoco se detectan signos claros de sobreajuste, ya que las métricas de validación se mantienen cercanas a las de entrenamiento.

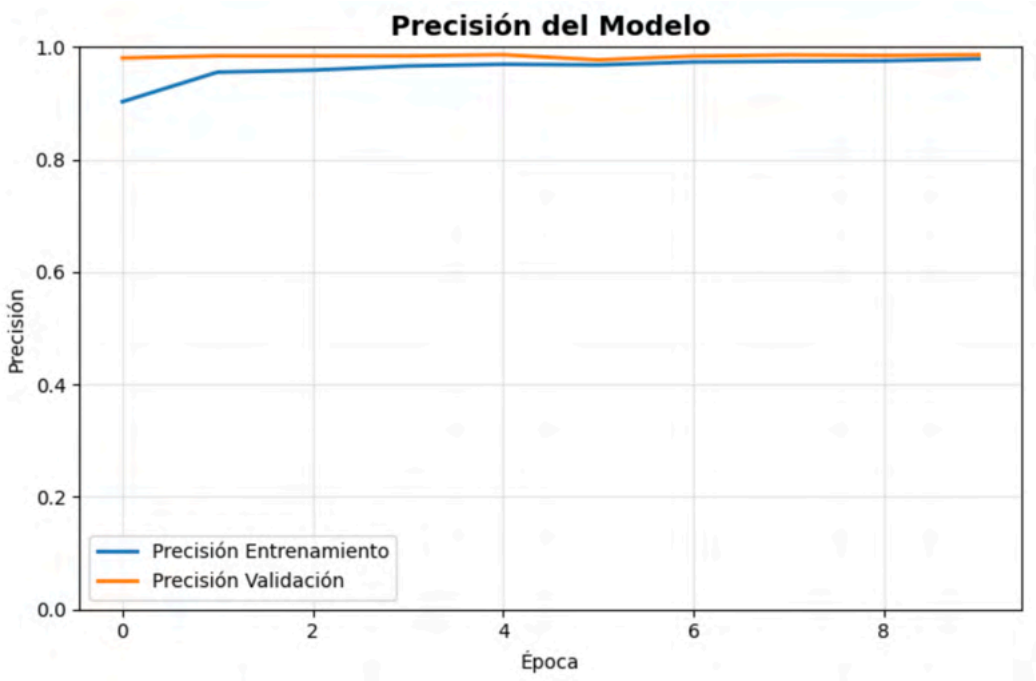


Figura 35 Evolución de la precisión del modelo.

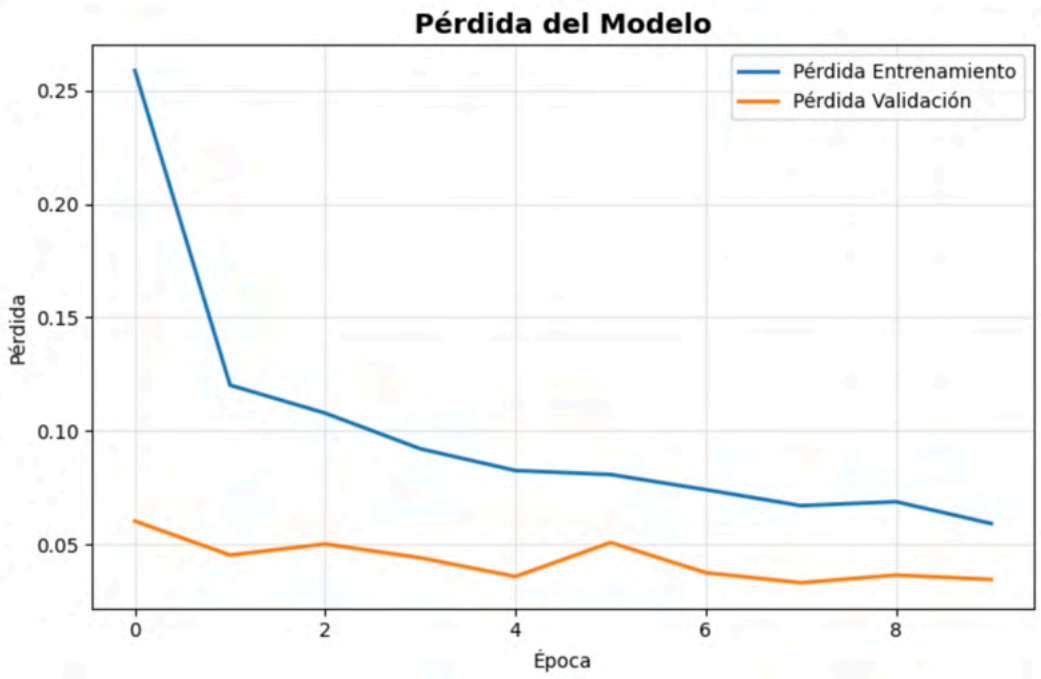


Figura 36 Evolución de la función de pérdida del modelo.

En conjunto, estos resultados confirman que el modelo desarrollado tiene buena capacidad de generalización, lo que lo hace apto para ser usado para predecir enfermedades con mucha precisión. Concretamente, al acabar el entrenamiento se obtiene una precisión en entrenamiento del 97,60% y en validación de 98,72%.

Una vez validado, el modelo se guarda en formato .keras, que conserva tanto la arquitectura de la red como los pesos entrenados y puede ser desplegado en un servidor web para clasificar nuevas instancias.

6.4 SERVIDOR WEB

Para que la aplicación desarrollada en este proyecto pueda acceder a este modelo de IA predictivo, es necesario que el modelo esté alojado en un lugar público y accesible. Para ello

se plantea el uso de un servidor web. De entre las múltiples opciones disponibles en internet, se selecciona la plataforma Render para crear un servidor donde alojar el modelo.

Render es una plataforma en la nube que permite a desarrolladores y equipos construir, desplegar y escalar aplicaciones, APIs y cargas de trabajo de inteligencia artificial de forma rápida y sencilla. Está pensada para facilitar el ciclo completo de desarrollo, desde el primer usuario hasta proyectos de gran escala, sin necesidad de gestionar servidores manualmente o usar Docker o Kubernetes.

Docker es una plataforma que permite crear, desplegar y ejecutar aplicaciones dentro de contenedores, los cuales son entornos aislados que contienen todo lo necesario para ejecutar una aplicación, como librerías, dependencias y configuraciones. Kubernetes, por otro lado, es un sistema de orquestación de contenedores que automatiza el despliegue, la gestión y la escalabilidad de aplicaciones dentro de contenedores Docker (Jesse & Castro, 2022).

Aunque estas herramientas son muy potentes y populares en infraestructuras de microservicios, Render ha sido diseñado para simplificar estos procesos. El uso de Docker y Kubernetes implica una mayor complejidad en la gestión de infraestructuras, lo cual puede ser innecesario para la mayoría de los desarrolladores que buscan soluciones rápidas y sencillas para desplegar sus aplicaciones. Render ofrece un entorno en el que no es necesario gestionar manualmente estos componentes, permitiendo a los usuarios centrarse en el desarrollo y la implementación sin preocuparse por la infraestructura subyacente.

En cuanto a los servidores web, Render permite alojar aplicaciones web escritas en prácticamente cualquier lenguaje y *framework* popular. También es posible desplegar imágenes Docker personalizadas. Cada servicio web en Render recibe un subdominio propio en onrender.com, aunque también se pueden añadir dominios personalizados. Los servicios web pueden comunicarse entre sí de forma segura a través de la red privada de Render, evitando exponer datos sensibles a la internet pública (Render, s.f.).

Para el despliegue del servidor web, se ha optado inicialmente por utilizar una instancia de tipo gratuita ofrecida por Render, adecuada para fines de prueba y validación. Esta instancia

proporciona 512 MB de RAM y 0.1 CPU. Sin embargo, este plan gratuito presenta algunas limitaciones. Tras periodos de inactividad, las instancias se detienen automáticamente y pueden requerir de varios segundos para volver a activarse en la siguiente petición. Además, no se admite acceso SSH, escalado automático ni almacenamiento persistente. Estas características son perfectamente válidas para el contexto actual del proyecto, en el que el objetivo principal es validar el funcionamiento del servidor y su integración con la aplicación móvil. No obstante, para un despliegue en entorno de producción, especialmente si se requiere ofrecer un servicio estable, de baja latencia y con disponibilidad continua, sería necesario optar por una instancia de pago que ofrezca mayores recursos de cómputo, persistencia de datos y soporte para escalado automático.

El servidor web que permite acceder al modelo de IA ha sido desarrollado de forma ligera usando *Flask*, un *microframework* de Python muy adecuado para construir APIs REST. Todo el proceso de despliegue se ha gestionado a través de un repositorio de GitHub con la siguiente estructura (figura 37):

- modelo_vid_compatible.keras → Modelo de IA entrenado y guardado.
- servidor_modelo.py → Código del servidor *Flask* que expone la API de predicción.
- requirements.txt → Lista de librerías necesarias para el entorno de ejecución.
- render.yaml → Fichero de configuración que define cómo Render debe desplegar el servicio.

Para el fichero requirements.txt, se incluyen las dependencias *flask*, *tensorflow*, *pillow* y *numpy* para el correcto funcionamiento del servidor.

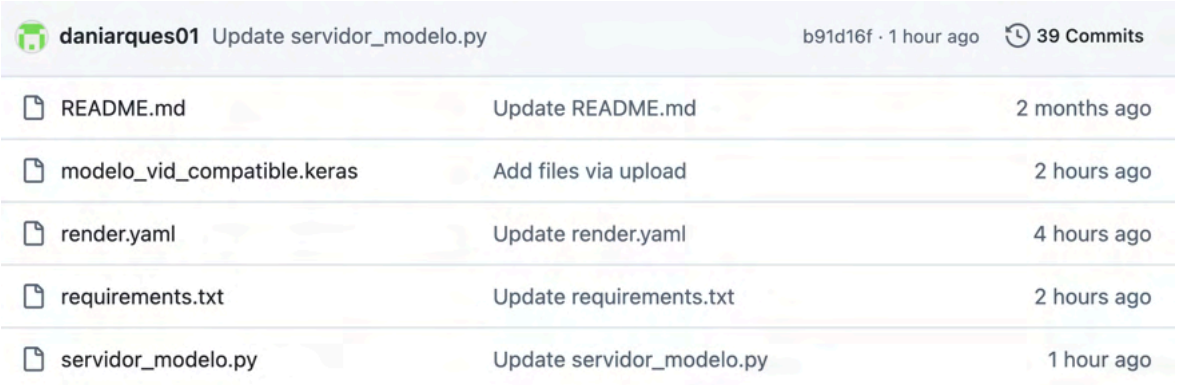


Figura 37 Estructura del Git.

El fichero render.yaml configura el despliegue como un servicio web que ejecuta directamente el script servidor_modelo.py y que escucha en el puerto 1000.

El código del servidor_modelo.py define dos rutas principales. La ruta raíz (/) devuelve un mensaje para comprobar que el servidor está en funcionamiento. En cambio, la ruta /predict recibe una imagen enviada (en este caso por la aplicación móvil), la procesa y devuelve la clase predicha por el modelo de IA de la siguiente manera:

1. Se recibe una imagen en la petición POST.
2. La imagen se guarda temporalmente y se procesa mediante la librería *tensorflow.keras.preprocessing.image*.
3. Se adapta la imagen al formato de entrada del modelo, es decir, a un tamaño de 224 x 224. No hace falta normalizar ya que esto ya lo realiza el propio modelo, con este paso configurado en el entrenamiento.
4. Se ejecuta la predicción con el modelo previamente cargado (modelo_vid_compatible.keras).
5. Se devuelve como respuesta en formato texto plano la clase predicha: "*Black Rot*", "*ESCA*", "*Healthy*" o "*Leaf Blight*".

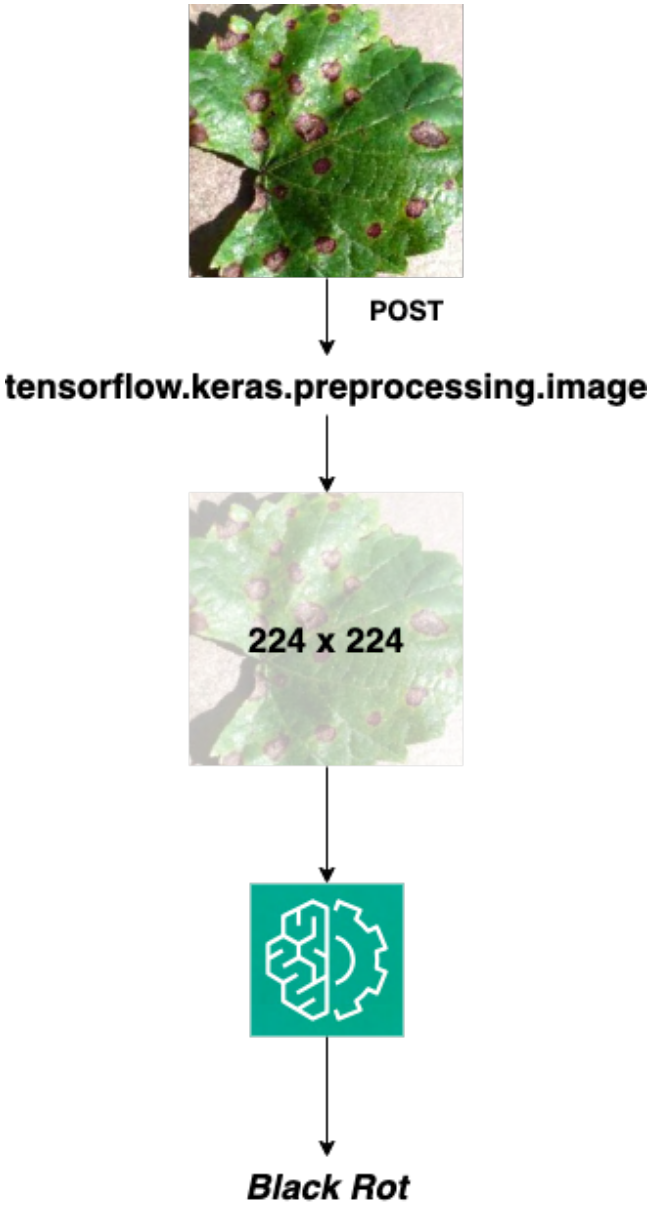


Figura 38 Diagrama de flujo del servidor.

El código que realiza estas acciones es el siguiente:

```
@app.route('/predict', methods=['POST'])
def predict():
    if not MODEL_LOADED:
        return jsonify({"error": "Modelo no disponible"}), 500

    if 'imagen' not in request.files:
```

```
        return jsonify({"error": "No se recibió ninguna imagen"}),
400

    img_file = request.files['imagen']
    img_path = "temp.jpg"
    img_file.save(img_path)

    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)

    pred = model.predict(img_array, verbose=0)
    pred_class = clases[np.argmax(pred)]

    if os.path.exists(img_path):
        os.remove(img_path)

    return pred_class
```

Una vez desarrollado y probado el servidor web localmente, se ha procedido a su despliegue automático en la nube mediante la plataforma Render. El proceso consiste en vincular el repositorio de GitHub que contiene el código del servidor, el modelo .keras, y los ficheros de configuración necesarios (requirements.txt, render.yaml). Render detecta automáticamente el fichero render.yaml, que define el servicio a desplegar como un servidor web basado en Python y especifica el comando de arranque.

Durante el proceso de despliegue, Render genera un contenedor de ejecución donde instala todas las dependencias necesarias a partir del fichero requirements.txt, y posteriormente ejecuta el script servidor_modelo.py, que inicializa el servidor Flask y carga en memoria el modelo previamente entrenado. Finalmente, Render asigna de forma automática una URL pública para el servicio desplegado. En este caso, el servidor queda accesible en:

https://tfm-mit-server.onrender.com

Desde esta URL, la aplicación móvil puede enviar imágenes mediante peticiones POST a la ruta */predict*, obteniendo como respuesta el diagnóstico generado por el modelo de inteligencia artificial. Este proceso automatizado de despliegue garantiza una integración continua y facilita tanto la validación como la futura actualización del servicio sin necesidad de gestionar servidores de forma manual.

6.5 FLUJO DE LA APLICACIÓN

El flujo de funcionamiento de la aplicación se ha diseñado para ofrecer una experiencia sencilla e intuitiva. A través de una primera versión de interfaz intuitiva y de una interacción amigable, el usuario puede acceder de forma rápida a las principales funcionalidades: consulta de tareas, revisión de rendimientos y diagnóstico de enfermedades.

6.5.1 ESCENA INICIAL

Al iniciar la aplicación, se carga la escena principal en la que se presenta la vista de cámara en tiempo real, sobre la que se superponen los siguientes elementos:

- Una barra superior con el nombre de la aplicación (VineGuard).
- Un campo de texto (ZoneText) que muestra la zona o parcela actual, determinada por el módulo de posicionamiento GNSS. Este texto se mantiene en la misma posición de la pantalla y se actualiza automáticamente al cambiar de zona.
- Un panel de texto (panelResultado) inicialmente oculto, destinado a mostrar la información correspondiente en función de la acción realizada.
- Una barra inferior con tres botones principales: tareas pendientes, rendimientos y diagnóstico, todo ello representado por iconos, tal y como se aprecia en la figura 39.

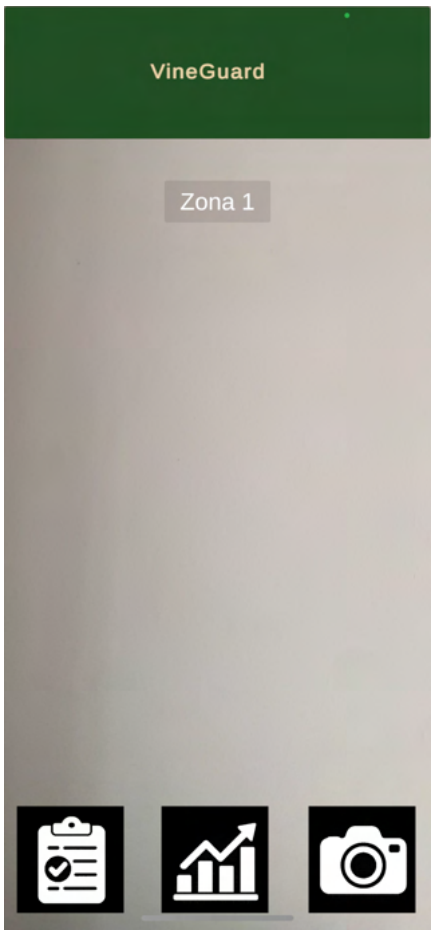


Figura 39 Estado de la aplicación.

Para definir las zonas, se utiliza un *script* en C# denominado GPSzonas.cs. Primero, se definen las zonas mediante arreglos de coordenadas geográficas, que incluyen la latitud y la longitud. Dependiendo del tipo de parcela, el proceso puede variar, pero en el caso más básico de parcelas rectangulares, se definen los cuatro puntos que delimitan la parcela.

A continuación, se verifica si el GNSS está habilitado en el dispositivo del usuario. Una vez confirmado que está habilitado, se inicia el servicio de localización, que en este caso particular tiene una precisión de 5 metros y se actualiza cada 10 metros.

Paralelamente, mediante una función, se espera hasta que el servicio GNSS esté completamente listo. Luego, se obtienen las coordenadas del dispositivo de forma continua. Posteriormente, se verifica en qué zona se encuentra el usuario, comparando sus coordenadas

con las previamente definidas para cada zona, utilizando un algoritmo para determinar si el usuario está dentro de los límites de la zona.

Finalmente, la zona identificada se muestra en la aplicación dentro del cuadro de texto ZoneText. Al salir de la aplicación, se liberan los recursos asociados al GNSS para evitar su uso innecesario. El código completo de este script puede consultarse en el Anexo 2.

Es importante remarcar que es el propio terminal el que obtiene las coordenadas del usuario, del mismo modo que lo hace para cualquier otra aplicación, generalmente mediante su antena GNSS (aunque existen otros mecanismos complementarios como el WiFi). Para que esto ocurra, hay que darle permisos a la aplicación, como se mostrará en el apartado 6.6. Una vez identificada la posición, se consigue clasificar por zonas gracias a los parámetros definidos en el script GestionBotones.cs.

6.5.2 LÓGICA DE INTERACCIÓN

La lógica de interacción está completamente gestionada a través del script GestionBotones.cs, que coordina el comportamiento de los botones y del panel de información. Cuando el usuario pulsa uno de los botones, ocurre lo siguiente:

1. Si el botón ya estaba seleccionado, la acción se deselecciona y el panel de información se oculta, quedando en el estado de la figura 39.
2. Si se selecciona un nuevo botón, se activa el panel y se muestra la información correspondiente a la funcionalidad de dicho botón (rendimientos, tareas o diagnóstico), como se puede ver en la figura 40.

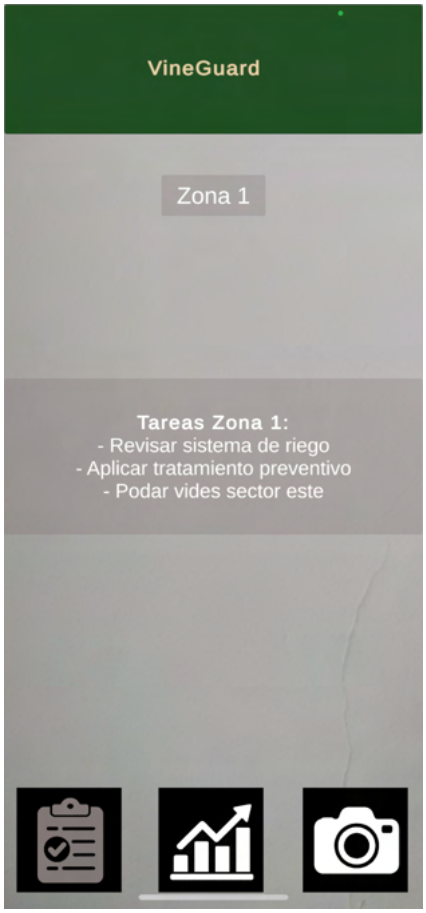


Figura 40 Estado de la aplicación al seleccionar un botón.

Concretamente, el comportamiento específico de cada botón es el siguiente:

Tareas pendientes: muestra en el panel las tareas planificadas para la zona actual. El contenido se adapta automáticamente en función de la zona mostrada en *ZoneText*, permitiendo una experiencia adaptada a la parcela actual. Ejemplos de tareas pendientes pueden ser fertilizar cierto sector de la parcela o podar las vides de algún sector.

Rendimientos: muestra en el panel los rendimientos históricos estimados para la zona actual, facilitando al viticultor la consulta rápida de los resultados de campañas anteriores. Tanto las tareas pendientes como los rendimientos son leídos del script GestionBotones.cs donde han sido definidos, eligiendo la información a mostrar en base a las coordenadas

GNSS del terminal. Las unidades proporcionadas son toneladas por hectárea (ton/ha) para cada tipo de uva.

Diagnóstico: se captura una imagen en tiempo real desde la cámara del dispositivo y se guarda en la galería. Posteriormente, se envía en formato PNG a través de una petición POST al endpoint /predict del servidor web desplegado en Render. Mientras se realiza el envío y procesamiento, se muestra un mensaje de espera en el panel. Cuando la aplicación recibe la respuesta del servidor se actualiza el panel con el diagnóstico de clase recibido, que corresponde a una de las categorías definidas en el modelo de IA: *Black Rot*, *ESCA*, *Healthy* o *Leaf Blight*.

El código completo para esta sección se encuentra en el Anexo 3.

6.5.3 GESTIÓN DE ESTADO

El diseño del flujo incluye una lógica de selección exclusiva de botones en la que, en cada momento, solo puede estar activo un botón, y el panel se muestra únicamente cuando una acción está en curso. Esta lógica mejora la usabilidad, evitando la saturación visual y asegurando que el usuario siempre reciba información clara y focalizada. El uso del componente *panelResultado* como área dinámica de información facilita además la ampliación futura del sistema, permitiendo incorporar nuevas funcionalidades sin necesidad de rediseñar la estructura básica de la interfaz.

A modo de resumen, la figura 41 muestra un diagrama sobre el funcionamiento de la aplicación explicado en este apartado.

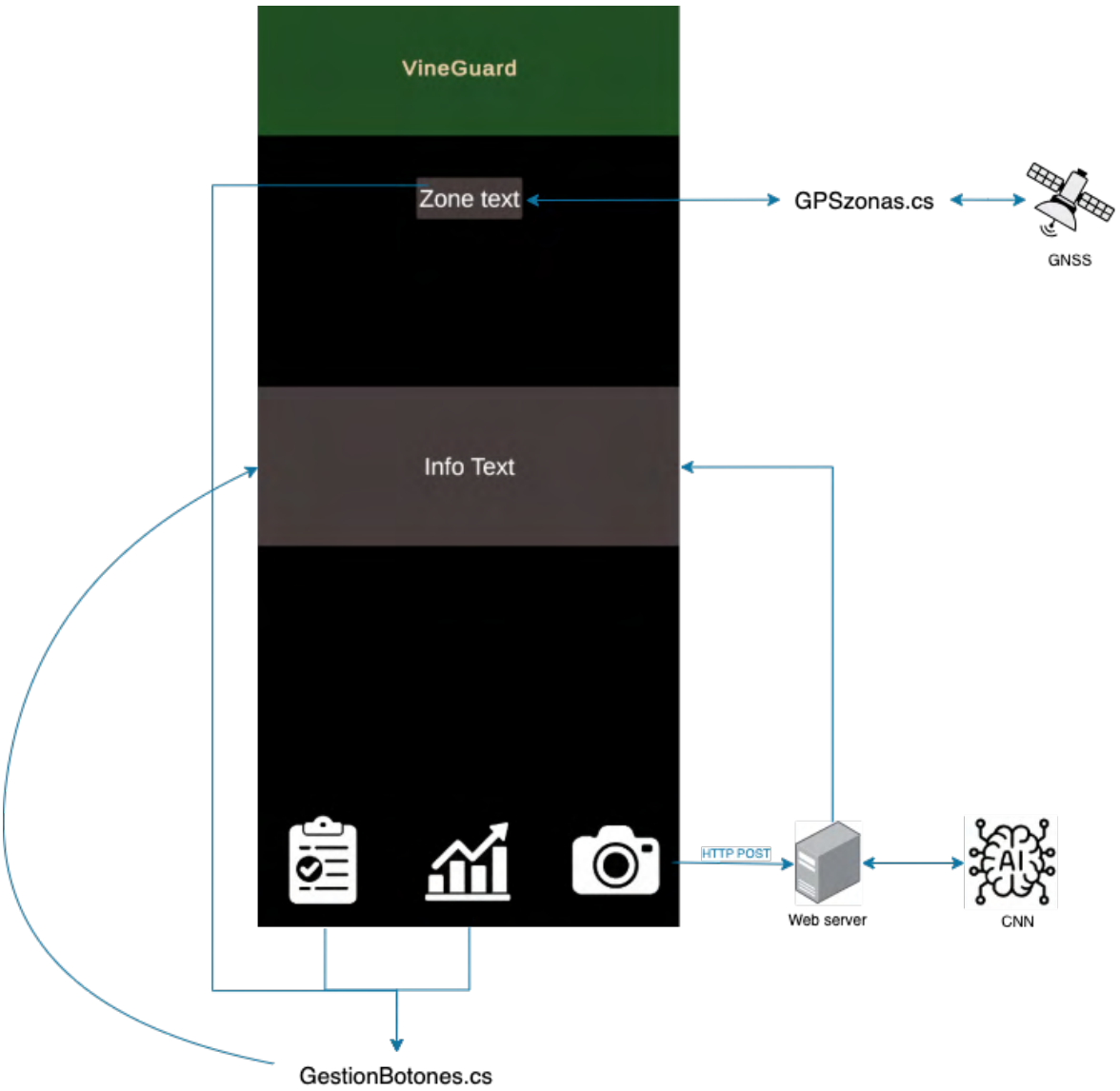


Figura 41 Diagrama funcional de la aplicación.

6.6 COMPILACIÓN Y DESPLIEGUE EN DISPOSITIVOS MÓVILES

Con el desarrollo de la aplicación finalizado en el entorno de Unity, es necesario proceder a su compilación y despliegue para dispositivos móviles, tanto en sistemas operativos Android como en iOS. A continuación, se detallan los pasos principales requeridos para compilar y ejecutar la aplicación en cada uno de los sistemas operativos.

6.6.1 EXPORTACIÓN PARA iOS

Para desplegar en los dispositivos Apple se requiere de un entorno macOS con Xcode instalado. Partiendo de este escenario, hay que instalar *iOS Build Support* desde Unity Hub en caso de no estar instalado. A continuación, en *Build Profiles* hay que cambiar de plataforma a iOS (figura 42), y en *Player Settings* hay que activar los permisos necesarios, que en este caso son los de la cámara y de la localización (figura 43), además de poder configurar las versiones o el identificador.

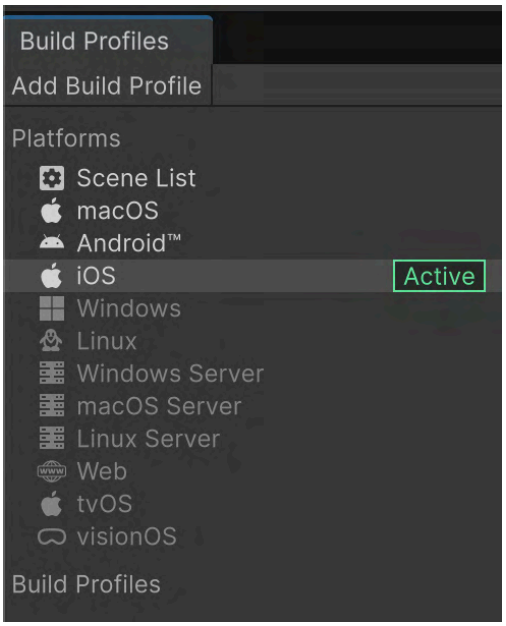


Figura 42 Build Profiles en Unity.

Figura 43 Permisos activados en los ajustes de Player.

Después ya se puede hacer Build and Run, y se genera un .xcodeproj que se abre en Xcode. Allí, se configura el equipo y la cuenta de desarrollador de Apple, se establecen los perfiles de aprovisionamiento y se selecciona el dispositivo objetivo. En ese punto, ya se puede compilar el proyecto en ese dispositivo y lanzar la aplicación.

6.6.2 EXPORTACIÓN PARA ANDROID

Para compilar la aplicación en Android, es necesario instalar *Android Build Support* desde Unity Hub, que incluye el SDK, NDK y OpenJDK, necesarios para poder desplegar una aplicación en Android. A continuación, en los *Build Profiles* de Unity es necesario cambiar de plataforma y elegir Android, de la misma manera que se hace para iOS. Allí hay varios parámetros que se pueden configurar, cómo el *Package Name*, el mínimo nivel de API o permisos específicos, que en este caso coinciden con los explicados para los dispositivos con iOS (cámara y localización). Finalmente, al ejecutar un *Build and Run* en Unity se genera un archivo .apk que debe ser transferido al dispositivo Android para poder ejecutar la aplicación.

Capítulo 7. ANÁLISIS DE RESULTADOS

En esta sección se analizan los resultados funcionales y las implementaciones prácticas realizadas, con el objetivo de validar críticamente si el sistema desarrollado cumple con los objetivos planteados en las fases iniciales del proyecto. Con fines prácticos, las demostraciones se van a realizar en un dispositivo Apple, concretamente, un iPhone XR con iOS 18.5.

Primeramente, se aprecia el estado inicial de la aplicación:

Figura 44 Estado inicial de la aplicación en la Zona 1.

A continuación, se muestra el estado de la aplicación tras cambiar de zonas, ya que la aplicación detecta correctamente las coordenadas GNSS del terminal que está corriendo la aplicación móvil.

Figura 45 Estado inicial de la aplicación en la zona 3.

En este caso, y para facilitar las pruebas, se han definido 3 zonas que simulan 3 parcelas distintas dentro de Madrid. Tal y como se aprecia en la figura 46, la zona 1 es la delimitada por el rectángulo rojo, la zona 2 por el rectángulo verde y la zona 3 es todo el territorio restante. Las zonas se definen con el script GPSzonas.cs explicado en la sección 6.5, y cabe remarcar que no habría limitaciones para añadir más zonas o cambiar su geometría o tamaño, pero esto es solo un escenario de pruebas.

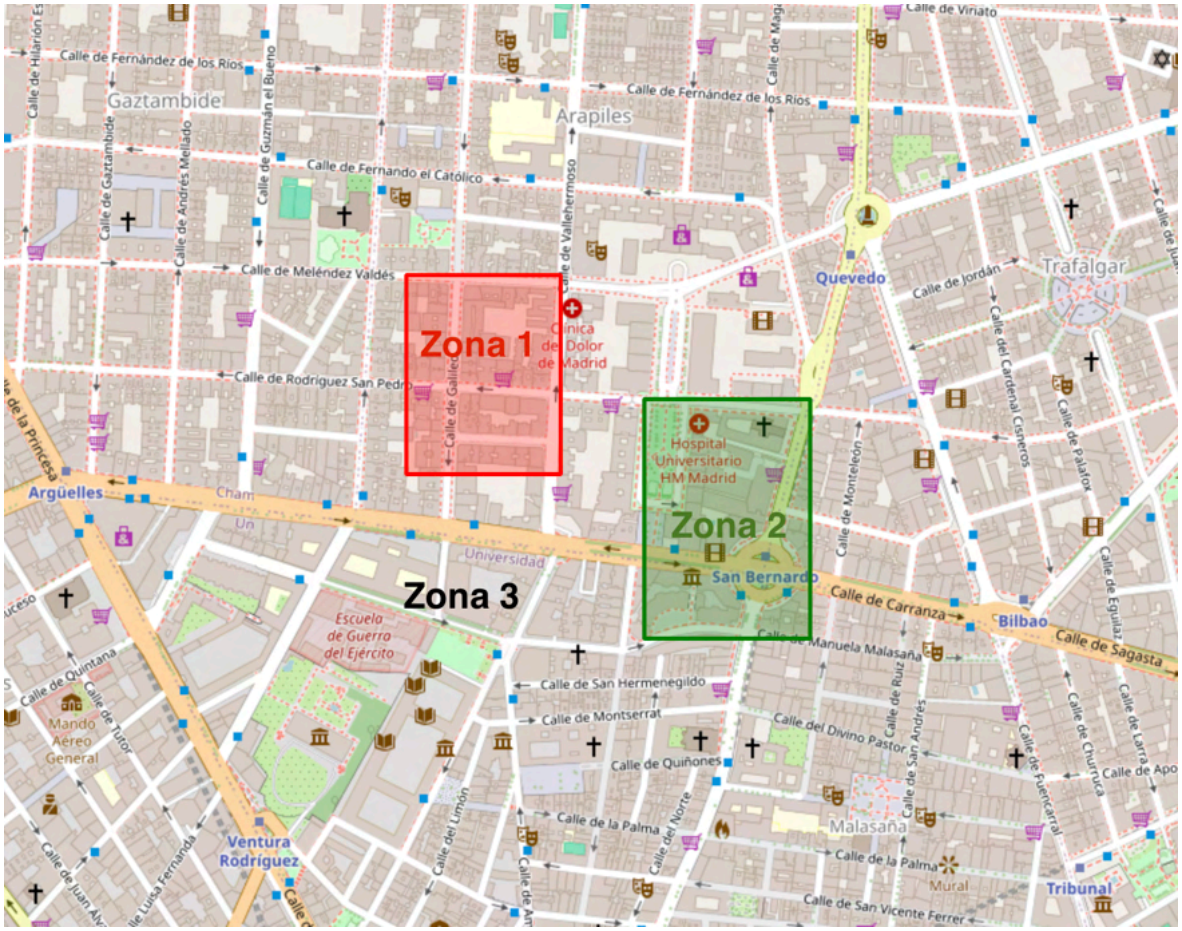


Figura 46 Zonas establecidas para pruebas.

Para cada una de las zonas, se muestran también las distintas tareas a realizar y los rendimientos de cada parcela (figuras 47 a 49). Estas tareas y rendimientos han sido introducidos a mano a modo de prueba para la validación del sistema, y podrían ser editados dentro del script GestionBotones.cs, explicado en el apartado 6.5.

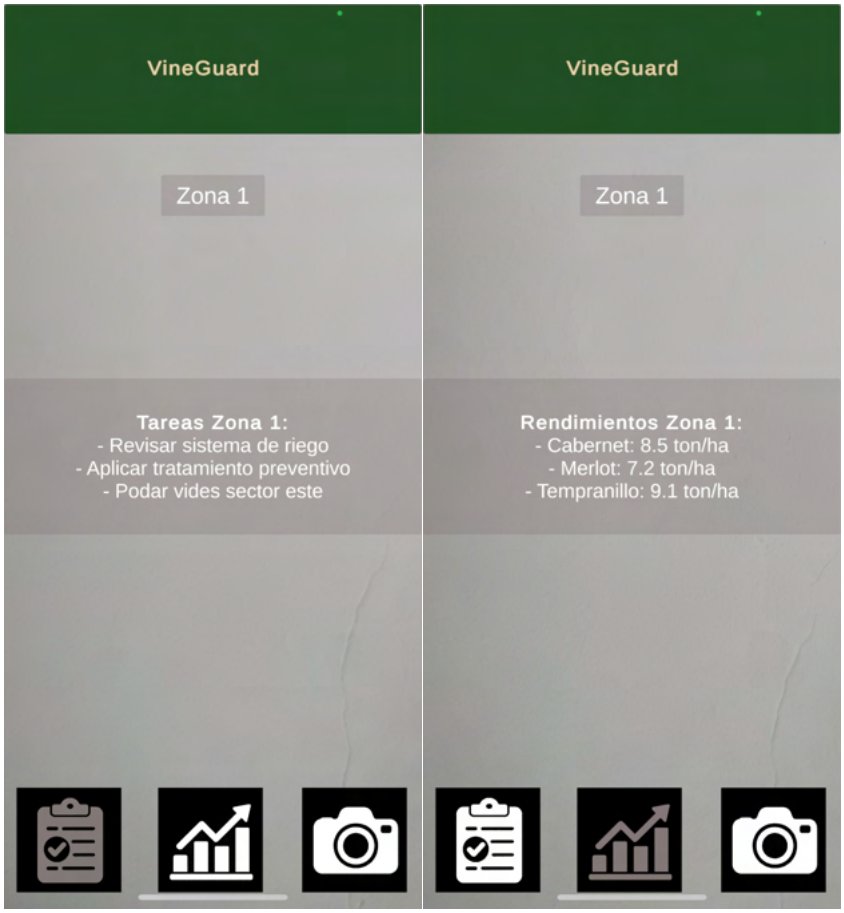


Figura 47 Tareas y rendimientos de la zona 1.

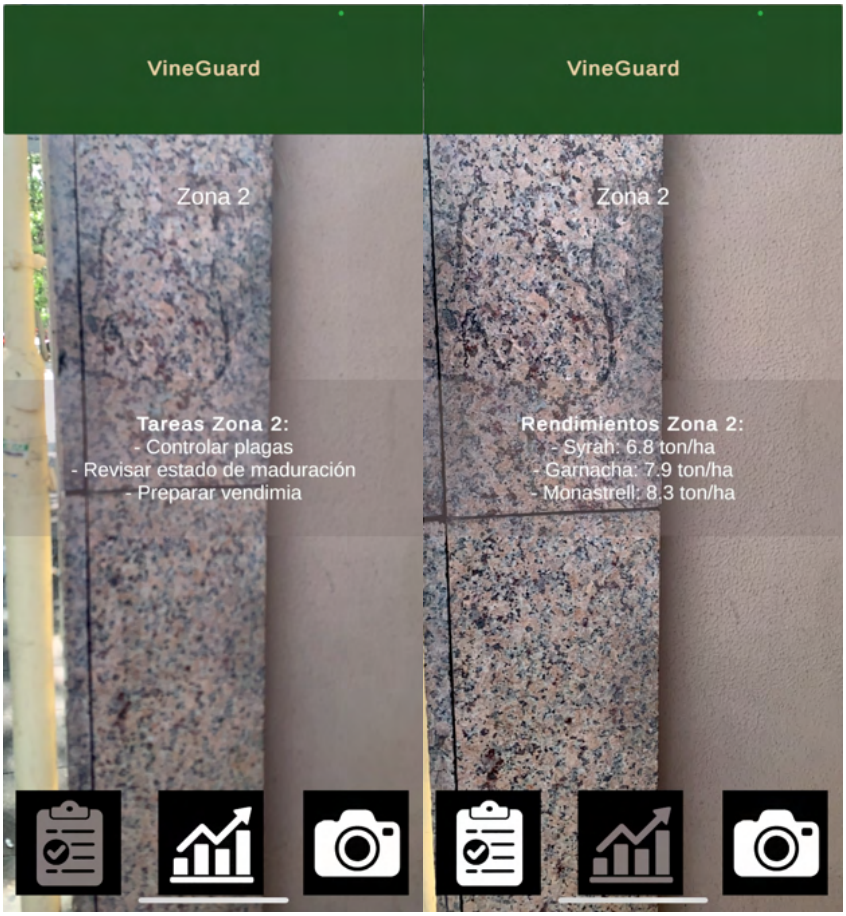


Figura 48 Tareas y rendimientos de la zona 2.

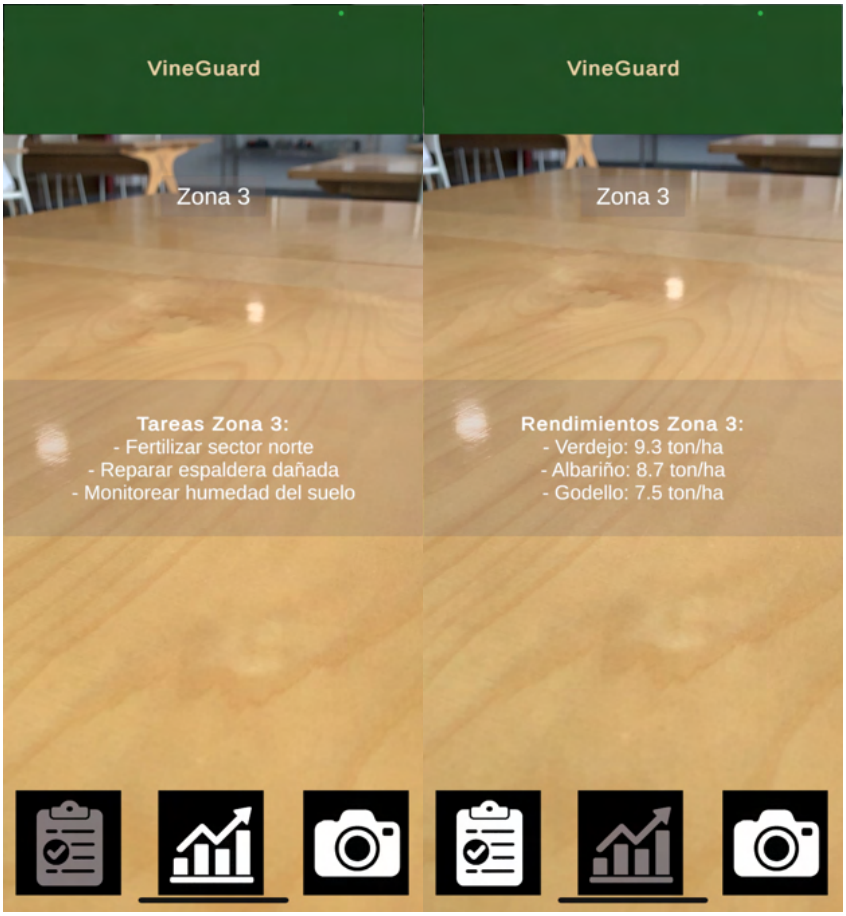


Figura 49 Tareas y rendimientos de la zona 3.

Para probar la funcionalidad de predecir el estado de salud de una vid, se realizan dos tipos de pruebas, una directa con las imágenes del dataset y otra con la aplicación detectando una hoja de la vid.

La primera de ellas es para probar el modelo entrenado de la forma más directa. En este caso, se usarán 4 imágenes distintas del dataset para ejecutar si predice bien cada una de las clases. Para hacer esta prueba, se hace uso de Postman, una herramienta que permite realizar peticiones HTTP de forma sencilla. En este caso, se utiliza el método POST enviando las imágenes directamente al *endpoint* del servidor y recibiendo como respuesta el diagnóstico generado por el modelo, sin intermediación de la aplicación móvil. Para realizar la comprobación, hay que configurar el método POST a la ruta /*predict* de la dirección en la que está alojada el servidor con la configuración que se aprecia en la figura 50. La

configuración requiere el formato *form-data*, que es adecuado para transmitir archivos a través de peticiones HTTP. La clave "imagen" es la esperada por el servidor *Flask* alojado en el servidor, y el valor es una imagen, que en este caso se corresponde a una planta sana, tal y como predice correctamente el modelo alojado en el servidor.

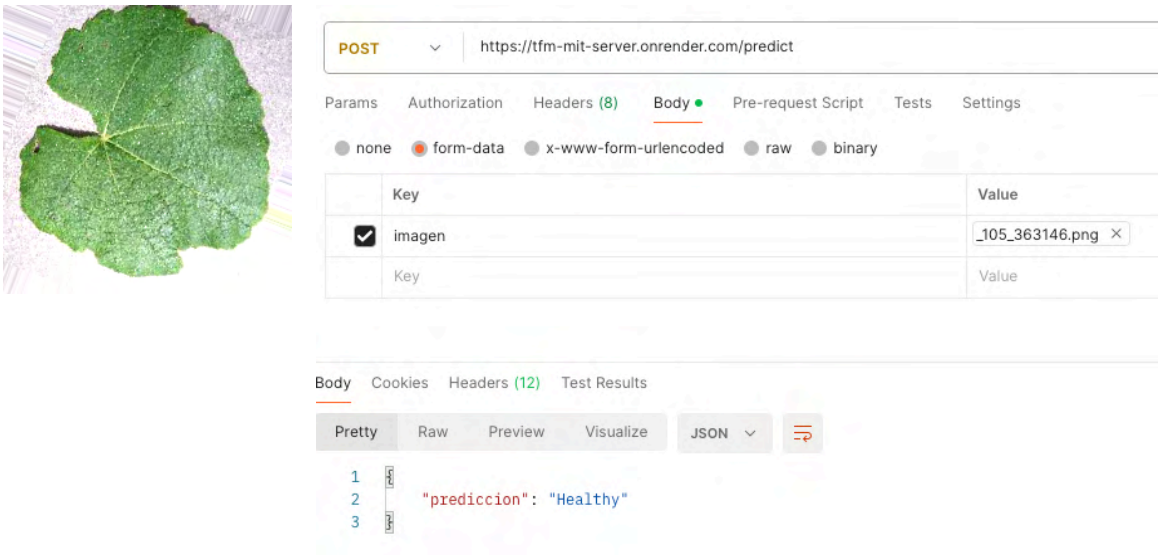


Figura 50 Configuración de Postman y prueba con planta sana.

Ahora se va a probar con un ejemplo de cada una de las 3 clases restantes, que coinciden con las 3 enfermedades que puede predecir el modelo (figuras 51 a 53).

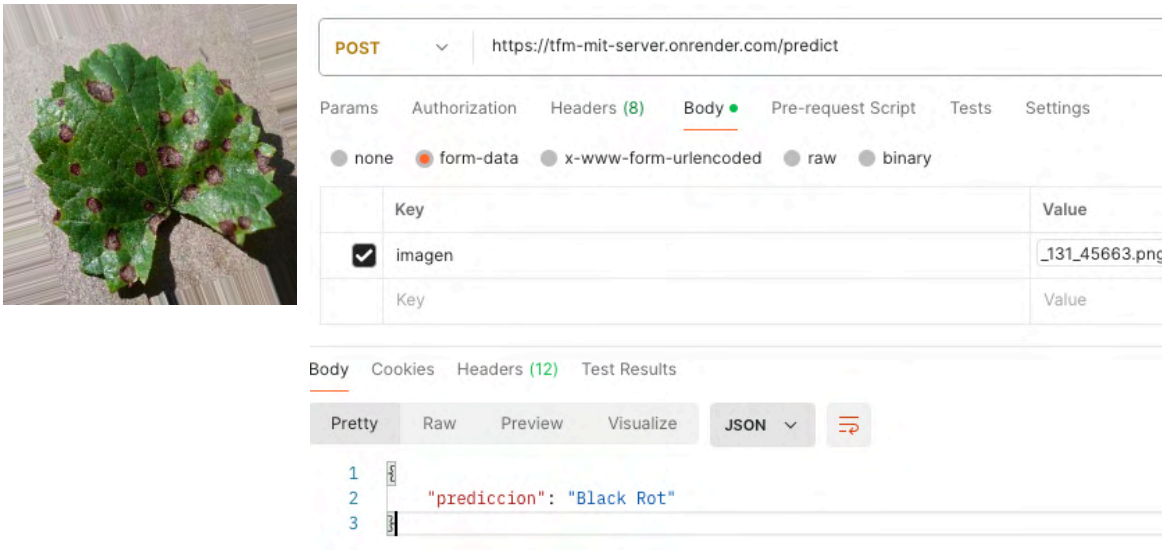


Figura 51 Prueba de planta con Black Rot.

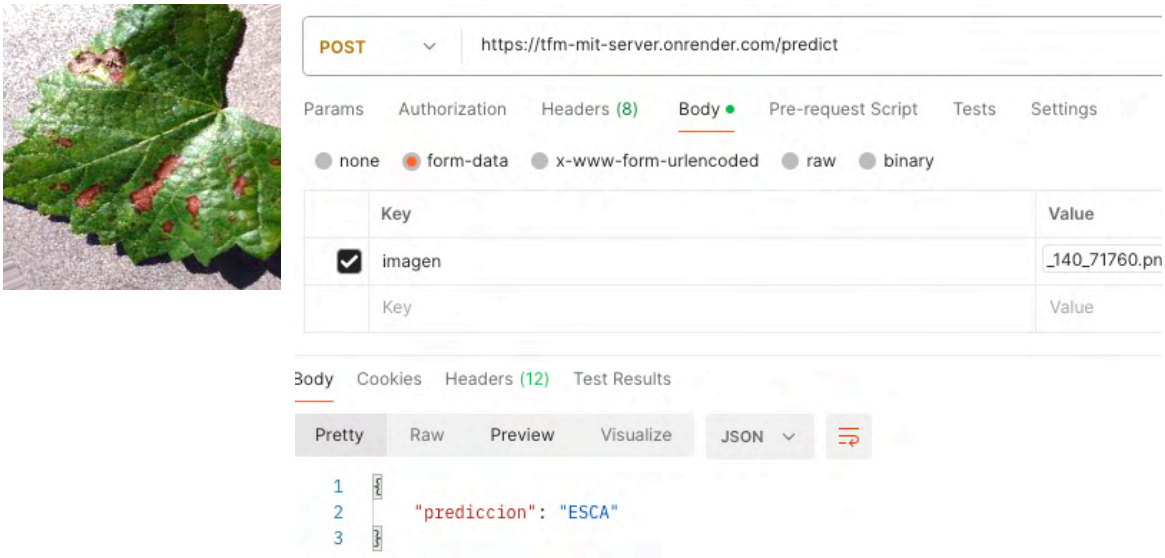


Figura 52 Prueba de planta con ESCA.

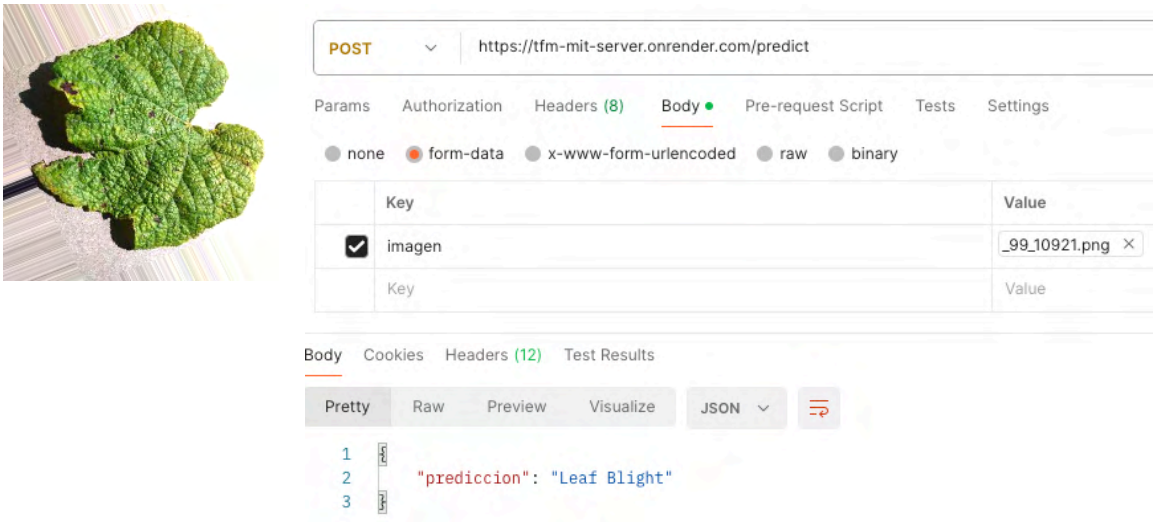


Figura 53 Prueba de planta con Leaf Blight.

Como se puede apreciar, el modelo funciona exactamente de la manera esperada. Por un lado, es capaz de predecir si la planta está sana o enferma y, además, en caso de que la planta esté enferma, predice correctamente su enfermedad con una precisión muy elevada (mayor al 95% según las curvas del entrenamiento).

Sin embargo, la prueba más interesante es directamente con la aplicación, ya que una imagen por Postman tiene una calidad muy buena, e incluso puede haber sido usada en el entrenamiento del modelo, por lo que la aplicación supone la prueba definitiva para ver si el modelo tiene capacidad de abstraerse o no. En este caso, por problemas de logística no ha sido posible trasladarse a un viñedo local, por lo que ha tenido que efectuarse con imágenes de hojas que son captadas por la aplicación.

Primero se efectúa la prueba con una planta sana. Para ello, se enfoca a una pantalla con una imagen de una de las hojas sanas de la base de datos.

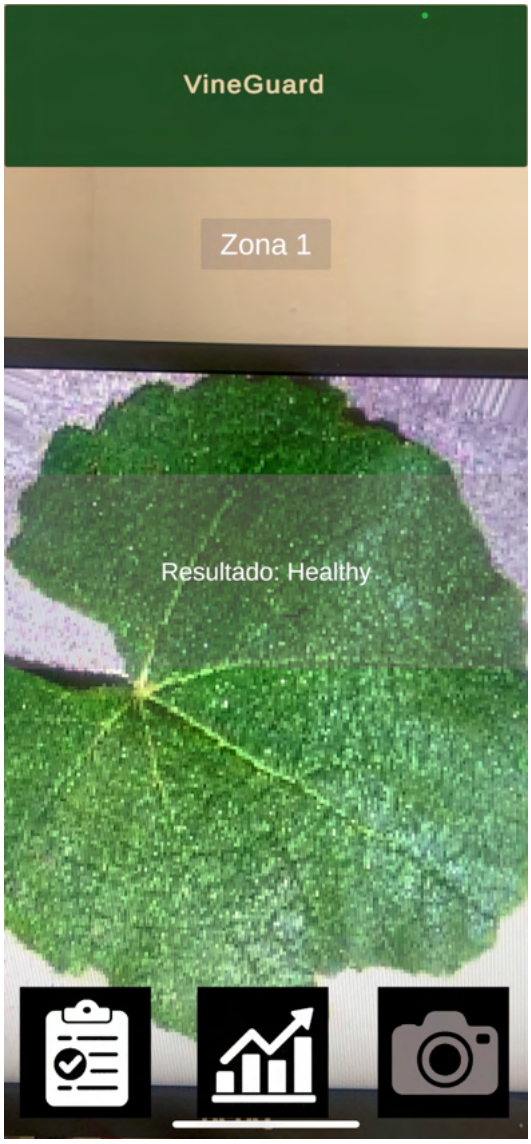


Figura 54 Resultado de planta sana.

Como se puede apreciar, el resultado es el correcto, así que el modelo está prediciendo bien con imágenes a las que no ha tenido acceso en el entrenamiento. Esto es así porque, al hacer una foto, la imagen capturada presenta variaciones naturales en iluminación, ángulo, encuadre y calidad, lo que garantiza que no sea una réplica exacta de las imágenes del conjunto de entrenamiento. Ahora analizamos los resultados de seguir el mismo procedimiento, pero con plantas enfermas.

Figura 55 Resultado de planta con ESCA.

Figura 56 Resultado de planta con Black Rot.

Figura 57 Resultado de planta con Leaf Blight.

Los resultados obtenidos en las figuras 55 a 57 demuestran que el modelo generaliza correctamente ante imágenes no vistas previamente, lo que evidencia una buena capacidad de abstracción y la ausencia de sobreajuste. Esto se traduce en una herramienta precisa que permite identificar las plantas enfermas y determinar su ubicación exacta, ofreciendo así un valioso recurso para la agricultura de precisión.

Capítulo 8. CONCLUSIONES Y TRABAJOS FUTUROS

8.1 CONCLUSIONES

Este proyecto ha consistido en el desarrollo de una aplicación móvil de realidad aumentada orientada al apoyo en la toma de decisiones agronómicas, integrando un modelo de inteligencia artificial para el diagnóstico del estado sanitario de vides. A través de la implementación de distintas funcionalidades, entre las que se incluyen la detección dinámica de zonas según coordenadas GNSS, la visualización de tareas y rendimientos por parcela y la clasificación automática del estado de salud de la vid, se ha logrado construir un sistema funcional y coherente con los objetivos iniciales del proyecto.

Los objetivos planteados al inicio han sido ampliamente cubiertos. Se ha diseñado una aplicación operativa, se ha entrenado y desplegado un modelo de IA con una precisión mayor al 95% y se ha validado la integración de ambos componentes en condiciones reales simuladas. La aplicación ha demostrado ser capaz de detectar correctamente las zonas definidas, adaptar su interfaz en función de la localización, y realizar predicciones fiables mediante el modelo de clasificación, incluso con imágenes captadas en tiempo real desde el dispositivo.

Entonces, se concluye el proyecto con una integración eficaz entre geolocalización, realidad aumentada, información contextualizada y un modelo predictivo de inteligencia artificial dentro de una aplicación móvil multiplataforma con un diseño intuitivo que sirve como una potente herramienta para el sector vitivinícola.

8.2 TRABAJOS FUTUROS

Aunque los resultados obtenidos han sido satisfactorios, existen diversas líneas de mejora y ampliación que podrían abordarse en trabajos futuros:

- **Conexión dinámica con fuentes de datos:** actualmente, las tareas y rendimientos asociados a cada zona están definidos estáticamente. Un desarrollo futuro clave sería la conexión con bases de datos o APIs externas, de forma que los datos agronómicos pudieran actualizarse automáticamente desde un sistema de gestión agrícola o plataforma externa, igual que definir nuevas zonas o modificar los límites de las parcelas.
- **Visualización de rendimientos por años:** otra mejora importante sería la implementación de un sistema que permitiera mostrar los rendimientos históricos por año, facilitando así el análisis de la evolución de la productividad a lo largo del tiempo.
- **Mejoras en ciberseguridad:** la comunicación entre la aplicación y el servidor podría reforzarse implementando protocolos de seguridad avanzados, como por ejemplo cifrado TLS, autenticación de usuarios o control de acceso basado en roles. Estas medidas serían especialmente relevantes en escenarios de uso real donde se manejen datos sensibles o privados.
- **Transición a un servidor profesional:** el servidor actual ha sido desplegado con fines de desarrollo. En una versión de producción, sería necesario migrar a una infraestructura de servidor profesional con garantías de disponibilidad, escalabilidad y soporte, lo que permitiría dar un servicio satisfactorio a múltiples usuarios simultáneamente y facilitar el mantenimiento a largo plazo.

En conjunto, el proyecto sienta unas bases sólidas tanto a nivel técnico como funcional, aunque también abre múltiples posibilidades de evolución futura con impacto real en la productividad y sostenibilidad del sector.

Capítulo 9. BIBLIOGRAFÍA

[1] 3D2cut. (s. f.). *3D2cut*. https://3d2cut.com/

[2] ABC Fichas. (s. f.). *La Neurona: Estructura y Función*. https://www.abcfichas.com/la-neurona-estructura-y-funcion/

[3] Al-Bayari, O., & Sadoun, B. (2007). Global Navigation Satellite System (GNSS). Princeton University.

[4] Andres, A. (2023). *Aprende qué es el Machine Learning y desentraña sus capas de abstracción*.

[5] Apple. (s. f.). *Xcode en Mac App Store*. https://apps.apple.com/es/app/xcode/id497799835

[6] Apple. (2025). *Apple Developer Program*. https://developer.apple.com/programs/

[7] Apple Inc. (2025). *Xcode*.

[8] Aprende Machine Learning. (2017). *Qué es overfitting y underfitting y cómo solucionarlo*. https://aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/

[9] Banachewicz, K., & Massaron, L. (2022). *The Kaggle Book: Data analysis and machine learning for competitive data science*.

[10] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer New York, NY.

[11] Bjorck, N., Gomes, C., Selman, B., & Weinberger, K. (2018). *Understanding Batch Normalization*.

[12] Comunidad de Madrid. (s. f.). *SIGPAC*. https://sigpac.mapa.gob.es/fega/visor/help/Manual%20de%20Usuario.html

[13] Cuadros Acosta, J. I. (2020). *Geek Electrónica AR: Prototipo de aplicación de realidad aumentada para dispositivos móviles Android, como apoyo en el aprendizaje de conceptos básicos de electrónica*.

[14] Elsherbiny, O., Elaraby, A., Alahmadi, M., Hamdan, M., & Gao, J. (2024). Rapid Grapevine Health Diagnosis Based on Digital Imaging and Deep Learning. *Plants*, *13*(1), 135. https://doi.org/10.3390/plants13010135

[15] GeeksforGeeks. (2025a). *Kernels (Filters) in convolutional neural network*. https://www.geeksforgeeks.org/deep-learning/kernels-filters-in-convolutional-neural-network/

[16] GeeksforGeeks. (2025b). *ReLU Activation Function in Deep Learning*. https://www.geeksforgeeks.org/deep-learning/relu-activation-function-in-deep-learning/

[17] GlobalViti. (s. f.). *GlobalViti*. https://globalviti.com/

[18] Goodfellow, I., Bengio, Y., & Courveille, A. (2016). *Deep Learning*.

[19] Google Play. (2019). *Ayuda de Google Play*. https://support.google.com/googleplay/?hl=es#topic=3364260

[20] Graniti, A. ; S. G. ; M. L. (s. f.). Esca of Grapevine: A Disease Complex or a Complex of Diseases. *Phytopathologia Mediterranea*, *39*.

[21] Gutiérrez, I. (2023). *Clasificación de imágenes con redes profundas*.

[22] He, Z. (2020). *Development of an augmented reality mobile application for museums*.

[23] Holopainen, T. (2016). *Object-oriented programming with Unity: Inheritance versus composition*. JAMK University of Applied Sciences.

[24] Hostinet. (2025). *Servidores VPS*. https://www.hostinet.com/servidores-vps/

[25] IAEco. (s. f.). *Deep Learning Colombia*.

[26] Izaurieta, F., & Saavedra, C. (2000). *Redes neuronales artificiales*.

[27] Jesse, B., & Castro, H. (2022). *Containerization and Orchestration in Cloud Computing*.

[28] Las Redes Neuronales. (2015). *Redes Neuronales*.

[29] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436-444. https://doi.org/10.1038/nature14539

[30] Lixandru, M., & Fendrihan, S. (2023). THE CONTROL OF BLACK ROT GUIGNARDIA BIDWELLI –A DANGEROUS FUNGAL DISEASE OF

GRAPEVINE. *Romanian Journal for Plant Protection*, *16*, 90-95. https://doi.org/10.54574/RJPP.16.11

[31] McCarthy, J. (2007). *What is Artificial Intelligence?*

[32] MediaMarkt. (2025). *Portátiles*. https://www.mediamarkt.es/es/search.html?query=port%C3%A1tiles

[33] Movistar. (2025). *Fibra Movistar*. https://www.movistar.es/fibra-optica/

[34] Mozilla Contributors. (2024). *Métodos de petición HTTP*. https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Methods.

[35] Mozilla Contributors. (2025). *What is a web server?* https://developer.mozilla.org/en-US/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_web_server

[36] Naciones Unidas. (s. f.). *Objetivos de Desarrollo Sostenible*. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[37] Nasteski, V. (2017). An overview of the supervised machine learning methods. *HORIZONS.B*, *4*, 51-62. https://doi.org/10.20544/HORIZONS.B.04.1.17.P05

[38] Piña-Rey, A., Ribeiro, H., Fernández-González, M., Abreu, I., & Rodríguez-Rajo, F. J. (2021). Phenological Model to Predict Budbreak and Flowering Dates of Four Vitis vinifera L. Cultivars Cultivated in DO. Ribeiro (North-West Spain). *Plants*, *10*(3), 502. https://doi.org/10.3390/plants10030502

[39] Rajarshi, M. M. K. G. (s. f.). Vineyard vigilance: Harnessing deep learning for grapevine disease detection. *Journal of Emerging Investigators*.

[40] Ramadhan, A., Raksa, F., Kurnia, K., Rizki, M., Octaviyani, S., & Angga, E. (2021). *WALL-FOLLOWING ROBOT NAVIGATION CLASSIFICATION USING DEEP LEARNING WITH SPARSE CATEGORICAL CROSSENTROPY LOSS FUNCTION*.

[41] Render. (s. f.). *Render Documentation*. https://render.com/docs

[42] Ribeiro, A., Bengochea-Guevara, J. M., Montes, H., Rodriguez, E., Fernández-Ortuño, D., & Andújar, D. (2023). Detection of damaged white grape bunches. *Proceedings of the 14th European Conference on Precision Agriculture*.

[43] Rojano, A., Salazar, R., Miranda, L., & Ojeda, W. (2021). *Algoritmo Adam en la inteligencia artificial*.

[44] Shinde, P. P., & Shah, S. (2018). A Review of Machine Learning and Deep Learning Applications. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 1-6. https://doi.org/10.1109/ICCUBEA.2018.8697857

[45] SIGPAC. (s. f.). *Visor SIGPAC*. https://sigpac.mapa.gob.es/fega/visor/

[46] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: a simple way to prevent neural networks from overfitting*.

[47] Talent.com. (2025). *Ingeniero telecomunicaciones: salario promedio en España*.

[48] Televitis. (s. f.). *Televitis*.

[49] Unity. (2025). *Products*. https://unity.com/es/products

[50] Unity Technologies. (s. f.-a). *Aprendiendo la interfaz – Manual de Unity*. Recuperado 18 de junio de 2025, de https://docs.unity3d.com/es/530/Manual/LearningtheInterface.html

[51] Unity Technologies. (s. f.-b). *Branding Trademarks*. https://unity.com/legal/branding-trademarks

[52] Unity Technologies. (s. f.-c). *Introducción a la Realidad Aumentada*. https://learn.unity.com/tutorials

[53] Vitality Learning. (2024). *Understanding Dropout: A Key to Preventing Overfitting in Neural Networks*.

ANEXO

Anexo 1

Antes de empezar el entrenamiento de la CNN se requiere la instalación de varias librerías de Python necesarias para ejecutar el código de entrenamiento. Son las siguientes:

- **numpy**

  Librería para el manejo de vectores y matrices multidimensionales, con funciones matemáticas optimizadas. Se utiliza para operaciones numéricas y para el tratamiento de imágenes como arreglos de datos.

- **Matplotlib**

  Herramienta de visualización gráfica. Se emplea para representar gráficamente ejemplos de imágenes y visualizar métricas de entrenamiento como la evolución de la pérdida y la precisión del modelo.

- **TensorFlow y Keras**

  Librerías principales para la construcción y entrenamiento de modelos de aprendizaje profundo. En concreto, se utiliza la API Keras integrada en TensorFlow para definir la arquitectura de la CNN, compilarla y entrenarla. También se utilizan sus módulos para la carga de datos, el preprocesamiento y la aplicación de aumento de datos.

Anexo 2

El código completo que gestiona las zonas definidas y su identificación dentro de la aplicación (GPSzonas.cs) es el siguiente:

```
using System.Collections;
using UnityEngine;
using TMPro;

public class ZonaGPS : MonoBehaviour
{
    // Definimos las zonas con las coordenadas de los vértices
    private Vector2[] zona1 = new Vector2[]
    {
        new Vector2(40.43295f, -3.71111f),
        new Vector2(40.43289f, -3.70881f),
        new Vector2(40.43092f, -3.71176f),
        new Vector2(40.43070f, -3.70896f)
    };

    private Vector2[] zona2 = new Vector2[]
    {
        new Vector2(40.43154f, -3.70757f),
        new Vector2(40.43113f, -3.70509f),
        new Vector2(40.42905f, -3.70782f),
        new Vector2(40.42884f, -3.70528f)
    };

    // Referencia al componente de texto para mostrar la zona
    public TextMeshProUGUI zonaText;

    // Intervalo de actualización en segundos
    public float intervaloActualizacion = 5f;

    void Start()
    {
        // Iniciamos la localización
        if (!Input.location.isEnabledByUser)
        {
            Debug.Log("GPS no habilitado.");
            zonaText.text = "GPS no disponible";
            return;
        }

        // Iniciar el GPS
        Input.location.Start(5f, 10f); // 5 metros de precisión, 10
metros para actualización

        // Esperamos hasta que la localización esté lista
        StartCoroutine(ActualizarUbicacion());
    }
```

```
    private IEnumerator ActualizarUbicacion()
    {
        // Esperamos hasta que la ubicación esté disponible
        while (Input.location.status == 
LocationServiceStatus.Initializing && Input.location.isEnabledByUser)
        {
            yield return new WaitForSeconds(1);
        }

        // Si no se pudo obtener la localización, mostramos un mensaje
        if (Input.location.status == LocationServiceStatus.Failed)
        {
            Debug.Log("No se pudo obtener la ubicación.");
            zonaText.text = "Error GPS";
            yield break;
        }

        // Bucle continuo para actualizar la ubicación
        while (Input.location.isEnabledByUser)
        {
            // Obtenemos las coordenadas actuales
            float lat = Input.location.lastData.latitude;
            float lon = Input.location.lastData.longitude;

            // Creamos un vector con las coordenadas obtenidas
            Vector2 coordenadasUsuario = new Vector2(lat, lon);

            // Comprobamos en qué zona está el usuario
            if (EstaDentroDeZona(coordenadasUsuario, zona1))
            {
                zonaText.text = "Zona 1";
            }
            else if (EstaDentroDeZona(coordenadasUsuario, zona2))
            {
                zonaText.text = "Zona 2";
            }
            else
            {
                zonaText.text = "Zona 3";
            }

            // Esperamos antes de la próxima actualización
            yield return new WaitForSeconds(intervaloActualizacion);
        }

        // Si salimos del bucle, detenemos la localización
        Input.location.Stop();
    }

    // Función que verifica si un punto está dentro de una zona
utilizando el algoritmo de ray-casting
    private bool EstaDentroDeZona(Vector2 punto, Vector2[] zona)
    {
        bool dentro = false;
```

```
        int j = zona.Length - 1;

        for (int i = 0; i < zona.Length; j = i++)
        {
            if (((zona[i].y > punto.y) != (zona[j].y > punto.y)) &&
                (punto.x < (zona[j].x - zona[i].x) * (punto.y -
zona[i].y) / (zona[j].y - zona[i].y) + zona[i].x))
            {
                dentro = !dentro;
            }
        }

        return dentro;
    }

    void OnDestroy()
    {
        // Asegurarse de detener el servicio de ubicación cuando se
destruye el objeto
        if (Input.location.isEnabledByUser)
            Input.location.Stop();
    }
}
```

Anexo 3

El código completo que gestiona el flujo de la aplicación (GestionBotones.cs) es el siguiente:

```
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;
using TMPro;
using System.IO;
using System.Collections;

public class GestionBotones : MonoBehaviour
{
    // Referencias a los botones
    public Button botonFoto;
    public Button botonTareas;
    public Button botonRendimientos;

    // Referencias a los paneles y textos
    public GameObject panelResultado;
    public TextMeshProUGUI resultadoTexto;

    // Referencia al script de zonas GPS
    public ZonaGPS zonaGPS;

    // Variables para la cámara y servidor
```

```
    private string serverUrl = "https://tfm-mit-
server.onrender.com/predict";
    private WebCamTexture camara;

    // Control de estado de los botones
    private Button botonSeleccionado = null;

    // Datos específicos para cada zona
    private string[] rendimientosZona = new string[3] {
        "<b>Rendimientos Zona 1:</b>\n- Cabernet: 8.5 ton/ha\n- Merlot:
7.2 ton/ha\n- Tempranillo: 9.1 ton/ha",
        "<b>Rendimientos Zona 2:</b>\n- Syrah: 6.8 ton/ha\n- Garnacha:
7.9 ton/ha\n- Monastrell: 8.3 ton/ha",
        "<b>Rendimientos Zona 3:</b>\n- Verdejo: 9.3 ton/ha\n- Albariño:
8.7 ton/ha\n- Godello: 7.5 ton/ha"
    };

    private string[] tareasZona = new string[3] {
        "<b>Tareas Zona 1:</b>\n- Revisar sistema de riego\n- Aplicar
tratamiento preventivo\n- Podar vides sector este",
        "<b>Tareas Zona 2:</b>\n- Controlar plagas\n- Revisar estado de
maduración\n- Preparar vendimia",
        "<b>Tareas Zona 3:</b>\n- Fertilizar sector norte\n- Reparar
espaldera dañada\n- Monitorear humedad del suelo"
    };

    void Start()
    {
        // Inicializar la cámara
        camara = new WebCamTexture();
        camara.Play();  // Activar cámara sin mostrarla

        // Ocultar el panel de resultados al inicio
        if (panelResultado != null)
        {
            panelResultado.SetActive(false);
        }

        // Configurar los listeners de los botones
        if (botonFoto != null)
        {
            botonFoto.onClick.AddListener(() =>
SeleccionarBoton(botonFoto, TipoBoton.Foto));
        }

        if (botonTareas != null)
        {
            botonTareas.onClick.AddListener(() =>
SeleccionarBoton(botonTareas, TipoBoton.Tareas));
        }

        if (botonRendimientos != null)
        {
            botonRendimientos.onClick.AddListener(() =>
SeleccionarBoton(botonRendimientos, TipoBoton.Rendimientos));
```

```
    }
}


// Identificar el tipo de botón
private enum TipoBoton
{
    Foto,
    Tareas,
    Rendimientos
}

void SeleccionarBoton(Button boton, TipoBoton tipo)
{
    // Si el botón ya está seleccionado, lo deseleccionamos
    if (botonSeleccionado == boton)
    {
        DeseleccionarTodos();
        return;
    }

    // Deseleccionar el botón anterior si existe
    if (botonSeleccionado != null)
    {
        ColorBlock colores = botonSeleccionado.colors;
        botonSeleccionado.image.color = colores.normalColor;
    }

    // Seleccionar el nuevo botón
    botonSeleccionado = boton;
    ColorBlock coloresNuevos = boton.colors;
    boton.image.color = coloresNuevos.selectedColor;

    // Mostrar el panel con la información correspondiente
    panelResultado.SetActive(true);

    // Ejecutar la acción correspondiente según el tipo de botón
    switch (tipo)
    {
        case TipoBoton.Foto:
            StartCoroutine(CapturarFotoYEnviar());
            break;
        case TipoBoton.Tareas:
            MostrarTareas();
            break;
        case TipoBoton.Rendimientos:
            MostrarRendimientos();
            break;
    }
}

void DeseleccionarTodos()
{
    // Ocultar el panel de texto
    if (panelResultado != null)
```

```
        {
            panelResultado.SetActive(false);
        }

        // Restaurar el color original del botón seleccionado
        if (botonSeleccionado != null)
        {
            ColorBlock colores = botonSeleccionado.colors;
            botonSeleccionado.image.color = colores.normalColor;
            botonSeleccionado = null;
        }
    }

    void MostrarTareas()
    {
        // Obtener el índice de la zona actual
        int indiceZona = ObtenerIndiceZonaActual();

        // Mostrar las tareas correspondientes a la zona
        resultadoTexto.text = tareasZona[indiceZona];
    }

    void MostrarRendimientos()
    {
        // Obtener el índice de la zona actual
        int indiceZona = ObtenerIndiceZonaActual();

        // Mostrar los rendimientos correspondientes a la zona
        resultadoTexto.text = rendimientosZona[indiceZona];
    }

    int ObtenerIndiceZonaActual()
    {
        // Obtener el índice de la zona actual
        int indiceZona = 2; // Por defecto Zona 3

        if (zonaGPS != null && zonaGPS.zonaText != null)
        {
            if (zonaGPS.zonaText.text == "Zona 1")
                indiceZona = 0;
            else if (zonaGPS.zonaText.text == "Zona 2")
                indiceZona = 1;
        }

        return indiceZona;
    }

    IEnumerator CapturarFotoYEnviar()
    {
        // Mostrar mensaje de espera mientras se procesa
        resultadoTexto.text = "Procesando imagen...";

        yield return new WaitForEndOfFrame();

        Texture2D foto = new Texture2D(camara.width, camara.height);
```

```
        foto.SetPixels(camara.GetPixels());
        foto.Apply();

        // Guardar la imagen
        string path = Path.Combine(Application.persistentDataPath,
"foto_capturada.png");
        File.WriteAllBytes(path, foto.EncodeToPNG());
        NativeGallery.SaveImageToGallery(path, "TFM-Vid",
"Foto_{0}.png");
        Debug.Log("Imagen guardada en galería: " + path);

        byte[] bytes = foto.EncodeToPNG();
        Destroy(foto);

        WWWForm form = new WWWForm();
        form.AddBinaryData("imagen", bytes, "foto.png", "image/png");

        UnityWebRequest www = UnityWebRequest.Post(serverUrl, form);
        yield return www.SendWebRequest();

        if (www.result == UnityWebRequest.Result.Success)
        {
            Debug.Log("Respuesta del servidor: " +
www.downloadHandler.text);
            resultadoTexto.text = "Resultado: " +
www.downloadHandler.text;
        }
        else
        {
            Debug.LogError("Error al enviar la foto: " + www.error);
            resultadoTexto.text = "Error al enviar: " + www.error;
        }
    }
}
```